SUPPORTING QUERY PROCESSING ACROSS APPLICATION SYSTEMS

Aspects of Wrapper-Based Foreign Function Integration

Klaudia Hergula
Research and Technology (Dept. FTK/A), DaimlerChryslerAG, HPC 0516, Epplestr. 225,70546 Stuttgart, Germany
Email: klaudia.hergula@daimlerchrysler.com

Gunnar Beck
Data Management, Marvin Consult,70197 Stuttgart,Germany
Email: g.beck@marvinconsult.de

Theo Härder
Dept. Of Computer Science (AG DBIS), University of Kaiserslautern,P.O. Box 3049, 67653 Kaiserslautern, Germany
Email: haerder@informatik.uni-kl.de

Key words: FDBS, WfMS, function integration, wrapper, heterogeneous query processing

Abstract: With the emergence of so-called application systems which encapsulate databases and related application components, pure data integration using, for example, a federated database system is not possible anymore. Instead, access via predefined functions is the only way to get data from an application system. As a result, the combination of generic query as well as predefined function access is needed in order to integrate heterogeneous data sources. In this paper, we present a middleware approach supporting this novel and extended kind of integration. Starting with the overall architecture, we explain the functionality and cooperation of its core components: a federated database system (FDBS) and a workflow management system (WfMS) connected via a wrapper. Afterwards, we concentrate on essential aspects of query processing across these heterogeneous components. Motivated by optimization demands for such query processing, we describe the native functionality provided by the WfMS. Moreover, we discuss how this functionality can be extended within the wrapper in order to obtain salient features for query optimization.

1. MOTIVATION

Most enterprises have to cope with heterogeneous system environments where different network and operating systems, database systems (DBSs), as well as applications are used to cover the whole life cycle of a product. Initial approaches primarily focusing on problems of data heterogeneity were federated database systems and multidatabase systems. So there exist adequate solutions for database integration even if there are still open questions [HST99, SL90].

But the database environment is changing now. While many enterprises had selected „their“ DBS and designed their tailored DB schema in the past, they are now confronted with databases being delivered within packaged software. In such cases, the database system and the related application are integrated, and an application programming interface, the so-called API, is the only way to access the data. Thus, a (generic) database interface is not supported anymore. In the following, we call systems realizing such an encapsulation concept *application systems*. One of the most frequently used application systems is, for example, SAP R/3 [SAP01], whose data can be accessed via predefined functions only. The same characteristics can be found in proprietary software solutions implemented by the enterprises.

As a consequence, pure data integration is not possible anymore, since “traditional” DBSs have to be accessed using a generic query language (SQL) whereas application systems only provide data access via predefined functions. Instead, a combined

approach of data and function access has to be achieved. Such scenarios can be encountered in many practical and/or legacy applications.

We consider an FDBS as an effective integration platform, since it provides a powerful declarative query language. Furthermore, it offers a large set of numerical processing functions as well as a broad range of scalability. Many applications are SQL-based to take full advantage of these properties. A query involving both databases and application systems includes SQL predicates as well as some kind of foreign function access. According to SQL99 [ISO99] such a reference may occur as a function, as a condition, or as a table. In our view, the most important case is the reference to a function as a table, strictly considered as an abstract table.

To implement such an extended kind of integration, we have developed an integration architecture consisting of two key components: an FDBS and a workflow management system (WfMS). The FDBS is responsible for the integration of data whereas the WfMS is used to implement a kind of function integration. As a result, the WfMS provides so-called global functions which are made available to the FDBS. Obviously, efficient query processing requires that these two components work together very closely. Such heterogeneous query processing reveals interesting aspects, since two completely different models – a data model and a function model – must be able to communicate and to work together.

In the remainder of this paper, we discuss basic questions about a smooth cooperation of these two components and we examine various implementation aspects of heterogeneous processing. For this purpose, we introduce our integration architecture in Sect. 2 by depicting the structure and the participating systems. Since the connection of FDBS and WfMS is a fundamental part of our architecture, we describe it in detail in Sect. 3. Next, we focus on heterogeneous query processing considering requirements of the functionality and its support in Sect. 4. In the remaining sections, we briefly review related work and summarize our ideas.

2. OVERALL ARCHITECTURE

The goal of our three-tier integration architecture is to enable the applications to transparently access heterogeneous data sources, no matter if they can be accessed by means of SQL or functions (see Fig. 1). Applications accessing the integrated data sources comprise the upper tier, and the heterogeneous sources represent the bottom tier. Due to space limitations, we focus on the middle tier, the so-called integration server, which consists of two key components: an FDBS achieving the data integration and a WfMS which realizes a kind of function integration by invoking and controlling the access to predefined functions. In our terms, function integration means to provide global functions combining functionality of one or more local functions [HH00]. The mapping from global to local functions is guarded by a precedence graph and it typically consists of a sequence of function calls observing the specific dependencies between the local functions. As a key concept of our approach, we use a WfMS as the engine processing such a graph-based mapping [HH00]. The workflow to be executed is a production workflow representing a highly automated process [LR00]. These two components are connected by an interface realized by means of a wrapper according to the draft of SQL/MED (Database Languages – SQL – Part 9: Management of External Data, [ISO00]). As a result, the WfMS provides so-called global functions used by the FDBS to process queries across multiple external data sources.

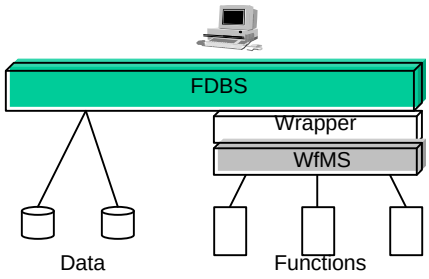

Figure 1: Integration architecture.

The global applications can access the integration server via an object-relational interface connecting them to the FDBS. The FDBS's query processor evaluates the global queries and those parts requiring foreign function access are handed over to the wrapper which activates the WfMS. The workflow engine performs the function integration by calling the local functions of the referenced application systems as specified in the predefined workflow process. The wrapper returns the result back to the FDBS where it is mapped to an abstract table. The remaining parts of the global query are processed by the FDBS, i.e., the query is divided into the appropriate SQL subqueries for the SQL sources. Eventually, the subquery results are further processed by the query processor, if necessary, and merged to the final result.

All issues of query processing in the FDBS are well explored [CRE87, MY95]. Therefore, we will focus on the new aspects of our architecture. How can we smoothly integrate calls of predefined functions into the overall query processing of an FDBS?

For this purpose, we have a deeper look at the wrapper-based connection between the FDBS and WfMS and its realization in the next section.

3. WRAPPER CONNECTION BETWEEN FDBS AND WFMS

Before discussing heterogeneous query processing in our architecture, we will examine the mechanism connecting the FDBS and the WfMS. Based on the assumption that the reader is familiar with the concept of a wrapper, we discuss how to map functions to tables within the wrapper and describe the tasks of the wrapper realizing the connection in detail.

3.1 Mapping Functions to Tables

The challenge of creating a wrapper between a relational FDBS and a WfMS is to bridge the conceptual differences between a data model and a functional model. A canonical mapping is to represent each function as a tuple of its input values and output values. Such a structure can be represented as a table, so the wrapper can enable access to the WfMS from the FDBS through abstract tables.

Functions can be classified in two categories. For each set of valid input parameter values, scalar functions return a single result tuple whereas relation-valued functions return a set of result tuples.

Our canonical mapping of relation-valued functions creates an abstract table containing a set of one or more rows of tuples containing input values and output values. A scalar function returns for each tuple of input values only a single output tuple. Therefore, the simple canonical mapping of scalar functions always creates a table with a single row.

3.2 Tasks of the WfMS Wrapper

In order to provide access to a WfMS for a relational FDBS, the wrapper must mediate between both systems, map the functional model to the data model, as well as provide support the communication between FDBS and WfMS. From the FDBS side, it must be able to accept a (partial) query regarding data access to the abstract tables. This query may include operators such as projection and selection using comparison predicates. Based on its knowledge about the abstract table representations, the wrapper prepares a local access plan. Such a local access plan consists of calls to WfMS process instances and may include wrapper-internal operations for requested functionality not natively supported by the WfMS. As usual, the optimization target is to push down as many predicates as possible to the WfMS to reduce data shipment.

During the execution of a local access plan, the wrapper invokes functions embodied by WfMS process instances. For each call, the wrapper derives input values from the query and receives result values from the WfMS. The input/output value pairs of each WfMS function call are processed by operations further explained in section 4. Finally, the query result set represented by an abstract table is returned to the FDBS query execution engine.

4. HETEROGENEOUS QUERY PROCESSING

In the following, we focus on basic technical aspects of heterogeneous query processing performed in the integration server. Starting with query processing requirements, we outline the functionality which should be provided by the wrapper. Afterwards, we describe the native functionality which is actually supported by the workflow engine. Based on our requirements we discuss how this functionality can be extended in the wrapper and how these extensions influence heterogeneous query processing and its optimization mechanisms.

4.1 Functional Requirements

In this section, we describe which operations should be supported by the wrapper by grouping them in core, base, and extended wrapper services. In addition, we explain why particular operations should not be addressed by the wrapper, but left to the FDBS instead.

Core functionality consists of all functions needed to map between abstract tables and WfMS function calls. If no additional functionality is available, each reference to an abstract table has to be realized by materializing and delivering the complete foreign table. For this reason, wrappers to be integrated into DBMS or FDBS processing should allow for basic optimization options, that is, they should enable push-down of selection and projection operations in the first place. Hence, *base functionality* on abstract tables should include projection to a subset of columns and selection of a subset of rows using any boolean combination of comparison predicates of columns with constants or a list of constants. Such comparison predicates contain an operator $q \in \{=, \neq, <, \leq, >, \geq\}$.

Core and base functionality could be *extended* by more advanced query execution operations such as

test of null values, subqueries regarding abstract tables, set comparison, and aggregation combined with grouping. Furthermore, the cartesian product of two abstract tables might be implemented within the wrapper in order to minimize communication caused by wrapper calls of the FDBS. On the other hand, join operators have been implemented in FDBSs in very efficient ways. Therefore, it should be critically evaluated whether or not a reimplementation of join functionality by the wrapper is justified.

Because wrappers receive only queries regarding abstract tables, there is no need to implement any functions beyond a subselect (such as union, except, intersect). Separate subselects are usually represented by independent queries to be combined by the FDBS. Subqueries regarding FDBS-managed tables should not be executed by the wrapper, since such functions can be handled better by an FDBS.

4.2 Native Query Support of the Wrapper

In the following section, we discuss the native functionality of the wrapper derived from the WfMS functionality. Based on the mapping of functions to tables described in Sect. 3.1, we now introduce the native mapping supported by the wrapper.

The wrapper is able to call a WfMS function and to receive its result set. In order to derive the input parameters needed for the function call, all input values are taken from the query. Hence, the input values for WfMS functions are specified by conjunctive combinations of comparison predicates between input parameters and constant values. Once, the set of function output tuples is received, these tuples can be extended by the input values of the function calls. The set of result tuples constructed in this process represents the extension of the abstract table. Finally, the result set is transferred to the FDBS query execution engine through abstract table queues.

In order to illustrate the mapped functionality, we describe the *function* $f_i(in_1, ..., in_m, out_1, ..., out_n)$ with in_j (j = 1 – m) and out_k (k = 1 – n) representing the input and output parameters as an SQL statement:

```
SELECT in1, ...,inm , out1, ..., outn
FROM fi_tab
WHERE in1θvalue1 AND ... AND inmθvaluem
```

All input parameters participate in comparison predicates, bound by AND operators. None of the output parameters is referenced in the where clause. Hence, this SQL statement precisely characterizes the expressiveness of our function evaluation on application systems performed through workflow processing.

4.3 Additional Query Functionality of the Wrapper

Describing the wrapper operations, we outline extensions of the supported functionality as introduced in Sect. 4.1. The core functionality for the mapping of WfMS functions to abstract tables is already covered by the native support as described in Sect. 4.2. Due to space limitations, we do not describe the operations in detail but rather focus on particular aspects caused by our approach and the resulting effects on query optimization.

4.3.1 Base functionality

In Sect. 4.1, base functionality was defined as projection and selection using boolean combinations of comparison predicates.

Projection of a subset of abstract table columns
Considering the projection operation, we have to distinguish two cases. In the first case, the projection includes columns representing the input parameters of workflow functions. As described in Sect. 4.2, the wrapper adds the input values to the abstract table. So the columns specified in the projection define which of the input parameter columns have to be added to the abstract table. In the second case, we consider the projection of columns representing the output parameters. The projection of all output parameter columns is equivalent to the result set returned by the WfMS. As a consequence, those output parameters not covered by the output specification must be removed by the wrapper. Of course, these two cases may appear in a single request.

Projection support within the wrapper can have a great impact on the optimization, since the size of a tuple and, as a consequence, the amount of data can be reduced substantially. Assume a tuple consisting of five attributes of comparable size. If the projection on a single attribute is supported, the size of the data returned is reduced to 20%.

Selection on rows of abstract tables
Selection on rows of abstract tables is based on a boolean combination of comparison predicates. The wrapper's core functionality supports conjunctive coupling of comparison predicates between the underlaying function's input parameters and constant values. The comparison operator must match the function semantics. In order to enhance the selection functionality, to minimize the number of wrapper calls, as well as the data to be shipped it is desirable to enhance the wrapper in three ways.

- Allow queries that do not bind all function input parameters to constant values by means of com-

parison operators. Such a functionality would enhance the overall query evaluation power.

- Process queries including comparison operators other than the ones determined by the function semantics. This includes comparison predicates regarding function output parameters, as well as additional comparison predicates regarding input parameters. An implementation of such functionality would reduce the query result set and therefore minimize the data volume to be shipped.
- Allow any boolean combination of comparison predicates. This functionality would minimize the number of queries passed by the FDBS.

First, we will explore how a wrapper can support all boolean combinations of comparison predicates. Any boolean expression can be transformed into disjunctive normal form (DNF). Disjunctive combinations of conjunctive brackets are derived by merging the results of all brackets. In the following, we will restrict ourselves to the evaluation of conjunctive brackets in DNF.

Within a conjunctive bracket, comparison operators regarding function input parameters that match the function semantics must be considered separately. Such predicates are used to extract the values needed for the function call. If not all input parameters are bound to values by comparison operators, those input parameters may contain any valid value. Assume the function is stateless and the input parameter domain is finite. Then such a partial conjunctive bracket represents the set of rows generated by merging result sets of separate function calls for each valid input value.

If the function is not stateless, the function's result depends on the order of function calls and therefore this method can not be applied. If an input parameter's domain does not have a limited number of elements, the number of required function calls might be infinite. In either case, the query must be rejected. All other comparison predicates within the conjunctive bracket can be applied to the intermediate result set generated by repeated function calls.

4.3.2 Extended Functionality

In this section, we outline functionality that we consider useful in a wrapper, but that is not required for a base implementation. The operations described here exploit processing knowledge about the implementation of WfMS functions primarily to minimize WfMS function calls or the data to be shipped to the FDBS. Due to space limitations, we only give an idea of how query processing is influenced when these operations are supported by the wrapper.

Test of null values: The implementation of a null-value test operator in a wrapper is desirable for minimizing WfMS calls from the wrapper and for minimizing the data volume transferred to the FDBS. Based on knowledge about the requested function, the wrapper might be able to answer a predicate without calling the WfMS. For instance, a column in an abstract table can not be null if this column is derived from a function's output and if it is known that, for each input value, the return value is not null.

Furthermore, the result of a not-null call is of type boolean. In most implementations, boolean is the most compact data type. Instead of returning the functions output values, only a single boolean value would be returned, minimizing the data shipment.

Subqueries regarding abstract tables: Considering subqueries, the distinction between correlated and uncorrelated subqueries is interesting. If the subquery is uncorrelated, it can be processed independently of the outer query block and has to be executed only once. The correlated case promises to be more challenging, since the subquery must be evaluated for each row of the outer table. This is only possible if the domain of this table is finite and if the domain elements can be enumerated. If the domain is not finite, it must be restricted by means of a predicate in the WHERE clause. In any other case, the query must be rejected.

Set comparison: In each case of set comparison, a set comparison predicate is followed by a subquery. Usually, comparison of sets requires much more data to be shipped between WfMS and wrapper and requires much more computation overhead than a subquery based on scalar comparisons. For this reason, query rewrite is very desirable. Most FDBSs use rule-based optimizers, but only a few of them can be extended to specific domains. The wrapper, however, may accomplish query rewrite based on abstract tables into WfMS function calls. Depending on the predicates applied to the abstract tables, equivalent WfMS functions of varying costs might be called. Hence, careful query rewrite offers considerable performance gains.

Aggregation combined with grouping: Extending the wrapper functionality to support aggregation combined with grouping, we have to differentiate between grouping functions applied to input parameters and those applied to output parameters.

The first case is supported only when the domain of the input parameter is finite and the enumeration of the domain elements is possible. The number of groups is known and, consequently, the number of WfMS functions calls.

If grouping is applied to output parameters, the grouping function has to be applied by the wrapper after retrieving the entire result set from the WfMS, no matter how many function calls are needed for

the set of qualified values. Again, the finiteness of the domain must be guaranteed.

The support of aggregation combined with grouping optimizes the overall query processing, since the number of tuples shipped to the FDBS is minimized.

5. RELATED WORK

Mediator- or wrapper-based approaches like Garlic [TS97] or TSIMMIS [PGW95] focus on general solutions and algorithms for integrating any kind of data source. Since in our case, all non-SQL sources are integrated by the WfMS, we have to consider only one single non-SQL source to be integrated: the WfMS. As a consequence, we can concentrate on a specific solution integrating the workflow system.

Furthermore, we focus on the integration of a functional interface at the FDBS side which has been discussed very early by [CS93], thereby demonstrating how references to foreign functions can be expressed in a query language. But they did not address the problem of limited access patterns, for which approaches like [FLM99] propose solutions by binding attributes in order to support queries on such data sources.

6. SUMMARY

In this paper, we have introduced an approach for the integration of heterogeneous data sources accessible via generic queries or predefined functions. We have described the components of our integration architecture introducing the FDBS, the WfMS, and its wrapper-based connection. Since FDBS and WfMS must cooperate when processing a global query, we have focused on heterogeneous query processing in our approach. First, we have pointed out the functional requirements of such a heterogeneous query processing system. We have described the operations which should be provided by the WfMS and the wrapper in order to support efficient heterogeneous query processing. Based on these requirements, we have shown how much of the required operations are supported natively by the WfMS functionality. Moreover, we have identified aspects which have to be taken into account when extending the native functionality by implementing additional operations within the wrapper. In addition, we have considered in what way the extended functionality can have an impact on query optimization.

REFERENCES

[CRE87] B. Czejdo, M. Rusinkiewicz, D.W. Embley: An Approach to Schema Integration and Query Formulation in Federated Database Systems, in: Proc. 3rd IEEE Int. Conf. on Data Engineering, Los Angeles, 1987, pp. 477-484.

[CS93] S. Chaudhuri, K. Shim: Query Optimization in the Presence of Foreign Functions, in: Proc. 19th Int. Conf. on Very Large Databases, Dublin, 1993, pp. 529-542.

[FLM99] D. Florescu, A. Levy, I.Manolescu, D. Suciu: Query Optimization in the Presence of Limited Access Patterns, in: Proc. ACM SIGMOD Conf. on Management of Data, Philadelphia, 199, pp. 311-322.

[HH00] K. Hergula, T. Härder: A Middleware Approach for Combining Heterogeneous Data Sources – Integration of Generic Queries and Predefined Function Access, in: Proc. 1st Int. Conf. on Web Information Systems Engineering, Hongkong, 2000, pp. 22-29.

[HST99] T. Härder, G. Sauter, J. Thomas: The Intrinsic Problems of Structural Heterogeneity and an Approach to their Solution, in: VLDB Journal 8:1, 1999, pp. 25-43.

[ISO99] ISO & ANSI: Database Languages - SQL -Part 2: Foundation, International Standard, 1999.

[ISO00] ISO & ANSI: Database Languages - SQL -Part 9: Management of External Data, Working Draft, September 2000.

[LR00] F. Leymann, D. Roller: Production Workflow: Concepts and Techniques, Prentice Hall, 2000.

[MY95] W. Meng, C. Yu: Query Processing in Multidatabase Systems, in: W. Kim (ed.) Modern Database Systems: The Object Model, Interoperability, and Beyond, ACM Press and Addison-Wesley, 1995, pp. 551-572.

[PGW95] Y. Papakonstantinou, H. Garcia-Molina, J. Widom: Object Exchange Across Heterogeneous Information Sources, in: Proc. 11th IEEE Int. Conf. on Data Engineering, Taipei, 1995, pp. 251-260.

[SAP01] SAP AG: SAP R/3, 2001; www.sap.com/solutions/r3/.

[SL90] A.P. Sheth, J.A. Larson: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, ACM Computing Surveys 22:3, 1990, pp. 183-236.

[TS97] M.Tork Roth, P. Schwarz: Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources, in: Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, 1997, pp. 266-275.