A Middleware Approach for Combining Heterogeneous Data Sources – Integration of Generic Query and Predefined Function Access

Klaudia Hergula
DaimlerChrysler AG
Research and Technology (Dept. FT3/EK)
P.O. Box 2360, D-89013 Ulm
klaudia.hergula@DaimlerChrysler.com

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

Abstract

With the emergence of so-called application systems which encapsulate databases and their applications, 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 focus on 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). Afterwards, we concentrate on the key problems of function integration by discussing query execution planning, precedence control of function execution, and parameter handling. In this context, we develop a light-weight description language based on XML for the global-to-local mapping of functions. In addition, we consider some important aspects of the execution model focusing on the interaction of the FDBS and the WfMS as well as the support of distributed transactions.

1. Motivation

Nowadays, most enterprises have to deal with heterogeneous system environments where different hardware components, network and operating systems, database systems (DBS), as well as applications are used to cover the whole life cycle of a product. In recent years, many research projects focused on various aspects of heterogeneity where interoperability between heterogeneous database systems was a key issue. As a result, researchers have developed concepts and prototypes of so-called federated database systems (FDBS) and multidatabase systems to integrate databases with different data models and weakly overlapping DB schemas. In the meantime, commercial products, so-called database gateways or database middleware [13], are already available. Accordingly, powerful solutions for specific aspects of database integration exist even if there are still open questions [6, 16].

But the database environment is changing now. While many enterprises had selected 'their' database system 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*. Frequently used application systems are, for example, PDM (product data management) systems like SAP R/3 [14], Metaphase [15] or ENOVIAVPM [3]. In the case of SAP, it is not possible for the application code to directly access by means of SQL the data stored in some relational database. Instead, SAP provides so-called BAPIs (Business APIs) by which the user can access the data via predefined functions. Similar to these commercial products, the enterprises often implement their own proprietary software solutions which can also be accessed by APIs only. Such approaches account for the fact that checking of integrity constraints as well as monitoring of security (authorization) are realized in the application code instead of using the database system's functionality. Often, conceptual schemas are not existing and expressive names are missing in order to save run time thereby enhancing the performance of the application. To avoid that changes in the DB schema may violate the consistency, integrity, and protection of the data, data access and invocation of functionality is allowed via APIs only.

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.

In the remainder of this paper, we present a middleware approach to integrate both data and functions. Starting from current integration approaches, we develop in Sect. 2 various requirements for functions to be integrated. In Sect. 3, we give an overview of our integration architecture introducing its major components. Afterwards, we focus on the integration of predefined function access and introduce the plan generation model and a mapping language based on XML to describe the function mapping in Sect. 4. The corresponding plan execution model is presented in Sect. 5, describing the software modules and their cooperation in more detail. After considering related work in Sect. 6, we finally summarize our ideas and give an outlook on future work in Sect. 7.

2. Requirements

In the following, we will examine the requirements to be met by extending data access of generic queries to the execution of predefined functions. We reveal the various demands by following the processing steps of a query. Starting at the client side and the given interface we observe the following difficulties the middleware has to cope with.

Interface to the client applications:

There are various possibilities to provide an appropriate client interface for generic query and/or predefined function requests. They may range from graphical user interfaces offering rich specification tools to conventional call interfaces requiring the input of linear query syntax. No matter how the queries are actually specified, they have to be transformed in an intermediate representation as a starting point for query evaluation. Obviously, there are two basic alternatives. Either, the access interface explicitly offers functions like methods of an object or stored procedures in an object-relational approach. Or, the functions are completely hidden behind a generic query. For instance, the function get_blue_cars could be replaced by the SQL query `SELECT * FROM all_cars WHERE color = blue`. Hence, the realized solution is dependent on the executive engine (e. g. a DBMS) as well as the anticipated interface at the client side, for example, visual query symbols or text fields for an explicit query input.

Query evaluation planning:

The result of query decomposition (performed in the FDBS) is an operator graph which embodies the overall query execution plan (QEP) including precedence relationships for the access of the participating data sources. In order to start with the query decomposition, the available functionality of the data sources (wrapper functionality) has to be known. For example, it is important to know whether a data source offers only sequential access to all objects (e. g. via a scan) or whether it accepts predicates for selective data access or even result order specification for sorted data delivery.

Query processing in the middleware:

Depending on the chosen alternative the query processor of the middleware must be able to extract those parts of a query which are issued as predefined functions of the integrated data sources. The selected query fragments have to be sent to the corresponding system components they can cope with.

Interfaces to the sources:

The middleware has to implement interfaces to the data sources to be integrated. These interfaces must overcome the heterogeneities of communication protocols as well as the heterogeneities regarding programming languages. Since the results are typically returned in different formats, the interfaces should translate them into the reference data model which is used inside the middleware. In addition, the middleware must be able to deal with the differences of error handling performed by the individual data sources. In summary, the middleware has to provide a homogeneous view across heterogeneous and distributed application interfaces.

Write access:

Another difficult issue even in the case of pure data integration is distributed write access to heterogeneous data sources. Support of a distributed transaction management is needed to guarantee the ACID principles [5]. Unfortunately, most of the integrated application systems are not able to participate in a distributed transaction processing protocol (two-phase commit protocol), since they do not support a 'precommit' state. So the middleware has to provide another solution, e. g. the definition of compensation actions.

Dependencies between/within application systems:

A global function may consist of several local functions which have to be called in a predefined order. In addition, the result of a local function may be used as input for another local function. So the middleware must be able to process a predefined execution order and to provide the required parameter values.

How should these requirements be mapped to a system architecture? As the key issue of our approach, we combine the services of a federated database system to provide data integration with the services of a workflow management system to handle platform and communication heterogeneities as well as precedence control of the overall processing. In the following sections, we will detail our approach and will also consider how well the listed requirements are met by our integration architecture.

3. Overall Architecture

This section gives an overview of our integration architecture and its components. We will propose a three-tier architecture with an FDBS and a WfMS constituting the core components.

The global applications providing graphical data access interfaces comprise the upper tier. The data elements to be combined are stored in different data sources managed in the bottom tier. These data sources can be composed of several types as described in Sect. 1, e. g. database servers and application systems. The goal of our integration architecture is to enable the global applications to transparently access the data sources, no matter if they can be accessed by means of SQL or functions.

The middle tier, the so-called integration server, consists of two essential components: an FDBS realizing the data integration, and a WfMS invoking and controlling the access to predefined functions. The applied workflow is a production workflow, i. e., it represents a highly automated process and integrates heterogeneous and autonomous application systems [11]. The global applications can access the integration server via an object-relational interface connecting them to the FDBS. The FDBS evaluates the global queries and functions and activates the WfMS if necessary. The interface connecting the FDBS and the WfMS may be realized in three different ways:

- by means of a wrapper according to the draft of SQL/MED (Database Language SQL – Part 9: Management of External Data, [9]);
- by implementing user-defined functions (UDF) as table functions, i. e., the UDF is used in the from clause of an SQL query;
- by calling the WfMS via stored procedures.

In every alternative, the result data derived by the workflow engine is kept using tables (relations) inside the FDBS. We will call such data *abstract table queues* (ATQs) in the following. The activated workflow engine realizes the function integration by calling the local functions of the integrated application systems as specified in the given workflow process. After completion the retrieved data is returned to the FDBS where it is subject to further processing (combination with other ATQs, output preparation, etc.).

Next, we will examine the roles the FDBS and WfMS may play in the architecture. We can outline a spectrum of possible constellations with the following extremes:

- *Pure FDBS solution*: The FDBS realizes both the generic data access and the predefined function calls. In this case, the application systems are accessed by wrappers which have to be provided for each application system. This solution allows to translate the heterogeneous ATQs derived from the heterogeneous data sources into a homogeneous data view very early, namely as a view of the FDBS data model. This means that we can take advantage of the whole functionality provided by the FDBS, above all its optimization capability and the supported operations to select, combine, and transform the retrieved data (e. g. projections, joins etc.). In addition, the query processing is very flexible since the queries can be processed dynamically at run time. Unfortunately, a separate wrapper for each application system has to be implemented to hide the existing heterogeneities.
- *Pure WfMS solution*: Considering the WfMS alternative, all data sources – application systems as well as databases – are accessed by the WfMS. This approach offers the advantage that the access to the heterogeneous systems is transparent to the developer of the integration server, since the WfMS deals with the heterogeneities concerning the communication protocols and interfaces. On the other hand, the processing of global requests is absolutely static because the workflow process has to be completely defined at build-time. Therefore, the only dynamical aspects in the run-time component are those concerning reactions to events.

Using both the FDBS and the WfMS, we want to explore a solution between the extremes outlined above. In the remaining sections, we assume that the WfMS is connected to the FDBS via a UDF used as table function which returns the workflow results as an ATQ. Further details will be given in Sect. 5.

In order to make clear how a global query is processed by the middleware, we illustrate each step separately (see Fig. 1). Starting with a global query (step ①) the FDBS's query processor has to evaluate which parts of the query can be processed by which data source (step ②). Those parts concerning functions are handed over to the UDF (step ③) which calls the WfMS (step ④). The workflow engine chooses the corresponding workflow process and executes the appropriate function calls (step ⑤). The results are returned to the FDBS as ATQs. The other parts of the global query are processed by the FDBS, i. e., the query is divided into the appropriate SQL subqueries for the data sources (step ⑥). The returned results are then further processed by the query processor and merged to the global result.[1]

All issues of query processing in the FDBS are well explored [2, 12]. Therefore, we will focus on the new aspects of our integration architecture. How can we smoothly integrate predefined function calls into the overall processing of an FDBS? For this purpose, we develop a description language by which the mapping of global functions to local ones can be specified. Afterwards, we present the execution model which finally realizes the integration of generic query and predefined function access.

[1] In particular, UDFs represent a proven mechanism available in object-relational DBMS. Our initial prototyping approach uses UDFs and is based on DB2 UDB and MQ Workflow.

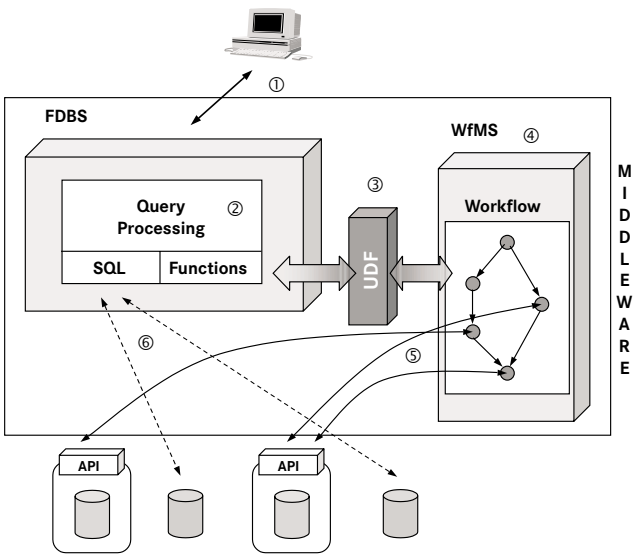

Figure 1. Integration architecture.

4. Plan Generation Model

After having presented the overall architecture, we now concentrate on a descriptive model to support the plan generation for global-to-local function mapping. Before describing the mapping itself we have a look on the tasks to be performed in a function mapping. First of all, the engine implementing the mapping must be able to call the local functions of the application systems. If there are dependencies between the local functions, a predefined execution order must be followed. Calling a function the engine must provide the required input parameters, that is, the parameters' data types have to be checked and value conversions must be carried out if necessary. Moreover, we have to cope with (n:m)-mappings regarding the parameters. In such cases, the parameters have to be adjusted by, for instance, merging or dividing them. Getting the output parameters returned by the local functions, the WfMS has to handle intermediate results which either are used as input for a local function or have to be transformed into the corresponding ATQ for the FDBS. Aside from these parameter-specific issues, the integration server must cope with different error codes and events returned by the application systems.

In order to be able to perform the described tasks, several types of information are needed. First, knowledge about the application systems to be integrated is required. Second, the global functions and the mapping itself must be defined. Moreover, the FDBS must know the input parameters for the UDFs and the resulting ATQs, because it has to translate and optimize the global query as well as to prepare the overall QEP. Focusing on the access to predefined functions the FDBS has to know which data may be retrieved via which ATQs. In addition, it must take into account the capabilities of the engine generating the ATQs, i. e. the WfMS. Does it only support a scan of the integrated data source, or is it possible to delegate operations like projections or selections? Or is it even possible to request a specific result order? Based on this information, the query processor optimizes and transforms the global query and, finally, generates a decomposable plan for the query execution. Those parts defined as a workflow schema will then be executed by the WfMS as a workflow instance.

In the following, we develop a mapping description which consists of two description documents based on XML (Extensible Markup Language, [20]): one illustrating the sources to be integrated and the other specifying the mapping to be performed. Since our intention is to keep the description as independent as possible from the implementation, the descriptions are not explicitly designed for the use with an FDBS and a WfMS. Moreover, access to the functions will be realized separately from the data integration, so we focus on the functions only. Thus, what we describe is a mapping between existing local functions and global ones which represent a kind of view on the local functions. In our architecture, the global functions serve as ATQs for the UDFs within the FDBS. After having explained how these descriptions look like, we additionally show to what extent the descriptions can be generated and where the interaction of the user is still needed.

4.1. System Description

The first part of our description contains information about the application systems to be integrated. Since the integration server resp. the WfMS realizes the access to the application systems, it is necessary to provide the required facts about the available functions and their signatures. Moreover, the location, i. e. the machine where the application system runs, the communication protocol and the executable program are essential items. Each application system is described in a single document which can be stored in a kind of repository. Since we want to use XML for the system descriptions we have to specify a DTD defining their structure. Examples can be found in [8].

Using the system description as metadata, the integration server should be able to call the functions of an application system and to handle the parameters without requiring further information.

In addition, we specify the global functions which should be provided in the global API in a separate XML document. These global functions can then be called by the global applications when connected to the integration server. After having completed the system description, the specification of the mapping itself can take place.

4.2. Mapping Specification

The next step in our descriptive model is the definition of the mapping language. The basic idea is to describe the mapping from global to local functions from their parameters' point of view by specifying dependencies between the parameters. Assume that there are no dependencies between the global functions, we can describe the mapping for each global function separately. Proceeding this way the complete mapping may be represented as a directed acyclic graph as illustrated in Fig. 2.

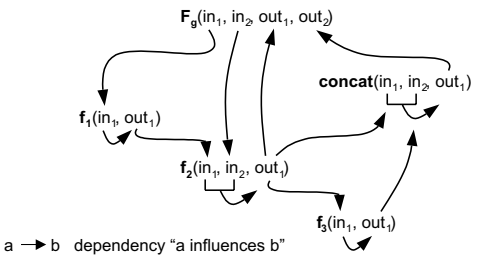

Figure 2. Mapping description by means of dependencies.

In this graph, the function mapping from one global function F_g to three local functions f_1, f_2, and f_3 is specified. Since the WfMS processes the function calls and returns the result for the FDBS, the global function F_g represents the result of the workflow process. The input of F_g is represented as the parameters in_1 and in_2, the output as out_1 and out_2. The input of F_g is used as input for the local functions f_1 and f_2. The output of the local functions is then mapped to a global output parameter (e.g. from f_2 to F_g) or is used as input for another local function (e.g. from f_1 to f_2). In addition, the function *concat* is needed to concatenate the results of f_2 and f_3 returning the value for the output parameter out_2 of F_g. The UDF finally has to transform the resulting list into an ATQ for the FDBS.

Note, if we consider the parameters and their dependencies, we can derive a directed acyclic graph where the parameters represent the nodes and the dependencies represent the edges. Right to that point, the definition of the execution order of the source functions is still missing. However, we do not have to define it explicitly if we refer to graph theory. A topological sort of the directed acyclic graph delivers a possible execution order. Since the dependencies define a partial order on the parameters, multiple topological sort orders are possible. This indicates that some parameters are independent and the functions can be processed in parallel. As a result, we are able to represent the parameter mapping as well as the execution order of the source functions in a coherent way.

So far, our approach only covers the case in which both systems are congruent w. r. t. parameter identifiers and data types. We still miss information needed to overcome the heterogeneities, e. g. operations concerning the parameters like converting their data types or combining them. Such operations are considered as functions and can be embedded in the mapping description (compare the function *concat* in Fig. 2). Thus, the representation remains homogeneous.

For the mapping language, we describe the dependency graph by enumerating the involved nodes, that is the function parameters, and the edges between them, that is the dependencies. In our approach, we use the current working draft of the linking language XLink (XML Linking Language, [21]). This linking mechanism extends the well-known link concept of HTML. In our paper, we will not explain the whole functionality of XLink, but consider the relevant parts for the mapping only. We make use of so-called extended links which allow the linking of any number of XML documents. What is so special about it is the fact that the linked documents don't need to provide an outgoing link. Hence, in our case we can define links between the parameters specified in the single system descriptions. An extended link consists of so-called locators and arcs where the locators define the participating sources and the arcs define the traversal behavior.

To describe the dependencies between the parameters, we use the extended link concept representing parameters as locators and dependencies as arcs. The DTD for the mapping description is shown in Fig 3. Here we can sketch only the basic idea, the full details can be found in [8].

```
<!ELEMENT function_map (node, dependency+)>
<!ATTLIST function_map
  xmlns:xlink CDATA              #FIXED "http://..."
  xlink:type  (sim|ext|loc|arc) #FIXED "ext">
<!ELEMENT node EMPTY>
<!ATTLIST node
  xmlns:xlink CDATA              #FIXED "http://..."
  xlink:type  (sim|ext|loc|arc) #FIXED "loc"
  id          ID                 #REQUIRED
  xlink:href  CDATA              #REQUIRED>
<!ELEMENT dependency EMPTY>
<!ATTLIST dependency
  xmlns:xlink CDATA              #FIXED "http://..."
  xlink:type  (sim|ext|loc|arc) #FIXED "arc"
  from        IDREF              #REQUIRED
  to          IDREF              #REQUIRED>
```

Figure 3. DTD for the mapping description.

Using these documents (system descriptions and mapping specification) as input, an appropriate engine is now able to process the integration of the predefined functions.

4.3. User Interaction and Automatic Generation

Going through the mapping specification process step by step, we discuss which parts need user interaction and

which can be generated automatically to some extent. The basis for the mapping specification are the system descriptions which, at best, are written down by the developers of the application systems. There are two possibilities: Either, there already exists a kind of repository where the application systems are described. In this favorable, but unlikely case, the descriptions have to be transformed into the appropriate XML document only. Or, there is no formal description yet, so the user has to describe the application system as an XML document. In any case, the user must define the API containing the global functions besides the description of the application systems.

When the system descriptions are completed, the user specifies the function mapping. This mapping specification as well as the system descriptions are then used as input for the build-time component of the WfMS. Based on the provided information a workflow process is built up within the build-time tool. The user can now determine further workflow-specific details. Based on this workflow process the build-time component generates the corresponding description which is given in the workflow process definition language (WPDL, [19]). This workflow process definition can then be used as input for the run-time component of a WfMS. In addition, the WPDL specification is automatically mapped to our XML-based mapping language generating an XML document. The description process is completed with the resulting XML document containing the mapping specification.

5. Plan Execution Model

In the following, we will describe the executive components of our architecture which realize the combined query and function access. As shown in Sect. 3, the integration server consists of two engines processing the global queries together. Since the required functionality to be provided by the FDBS is known from several approaches regarding algebra graphs and heterogeneous plan generation we will not go into this topic in further detail, but refer the interested reader to [17]. Instead, we will concentrate on the workflow run-time component, the connection between the FDBS and the WfMS, and finally have a look on the available support for distributed updates.

5.1. Workflow Run-Time Component

As described in the previous sections, the run-time component of the WfMS represents the executive engine for our function mapping. Based on the process description generated by the build-time component, the workflow engine takes the input parameters of the FDBS and starts calling the appropriate functions in the specified order. It also guarantees transparent access to the different platforms which includes heterogeneous communication protocols, different operation systems, and the varying representations of the data types in different programming languages. In addition, it must be able to cope with different kinds of error handling, e. g. exceptions or return codes. Unfortunately, most WfMSs do not support data type mapping like casts from integer to real. However, this functionality is needed, since the output parameter of one local function is often mapped to another function's input parameter of different data type. So we add another system to our architecture – we will call it the helper system – which provides those functions needed for the conversion of data types. Moreover, the helper system may also contain functions like *concat* introduced in Sect. 4.

5.2. UDFs Building the Bridge Between Relations and Functions

The implementation of the UDF is one of the most challenging parts in our architecture, since it has to represent a bridge between two systems supporting different representations of data. As we know, the FDBS stores its data in relations whereas the WfMS supports basic and semi-complex data types. So the UDF has to transform the results returned by the WfMS into corresponding ATQs for the FDBS. Considering a global function, the UDF has to provide the input parameter values for the WfMS and, in the opposite direction, the output parameter values for the FDBS. Each UDF creates for the corresponding global function an ATQ containing the output parameters as its attributes. For our example, the global function $F_g(in_1, in_2, out_1, out_2)$ is represented as the relation F_g containing the attributes out_1 and out_2. The input parameters are still used as an input parameters for our table function. The global function is then translated to the following SQL query:

```
SELECT out1, out2
FROM TABLE (Fg(in1, in2)) as Fg
```

As a result of the heterogeneous plan generation, the FDBS calls the UDF with the given input parameter values. The UDF then starts the workflow engine providing the values as input for the workflow process. After successful execution the resulting output is transformed to an ATQ containing the attributes listed in the select clause.

5.3. Transaction Management

Write access to data sources adds difficult problems to our integration architecture. The following issues have to be solved:

- When integrating data sources with overlapping schemas, dependencies between the sources may arise. In that case, integrity checks over several data sources

have to be realized. Therefore, a kind of global integrity control is needed.

- Another issue arises from creating global data objects by bringing together data extracts of several data sources. Often, these global objects are the result of views. Therefore, the well-known problems regarding updatable views may occur.
- The most important point when supporting updates is to guarantee the consistency of the data by providing a transaction management. In our case, we even need a kind of heterogeneous transaction management in order to be able to support distributed updates over heterogeneous data sources.

Here we only want to sketch some aspects related to transaction management.

If we want to support distributed updates in our integration server a distributed transaction management has to be provided. However, a distributed transaction management may be realized only if the participating systems support a two-phase commit protocol (2PC). We can expect such a functionality from most of the DBSs, whereas application systems are usually not designed to support a 2PC. As a result, we have to consider two points:

- How do we realize a distributed transaction management for the integration server, i. e. the WfMS?
- When the WfMS also does not support a 2PC protocol, how can we provide distributed transaction processing within the FDBS given the WfMS participates and controls the precedence flow of processing?

Analyzing the first problem, a conventional transaction management cannot be realized due to the missing 2PC support by the application systems. If we are not able, however, to initiate a rollback in the integrated systems we have to apply compensations. This means that we have to specify a compensation function for every function called by the WfMS. In the case that a participating system fails in a distributed update, the compensation functions have to be executed for those systems which have already finished their work successfully. This idea has been introduced in [10] for workflow systems and is called the concept of compensation spheres. A compensation sphere may consist of transactional as well as non-transactional activities. When mapping a global function to a workflow process, all activities contained establish a compensation sphere and, thus, a new kind of unit of work. As a result, we are able to support distributed updates.

The second issue we have to cope with is the fact that the WfMS itself does not support a precommit state to the outside. As a consequence, the FDBS cannot process a distributed update across the DBMSs and the WfMS. As long as there is only one source not supporting the 2PC there is another possibility to realize a distributed transaction. In that case, the FDBS sends a prepare-to-commit to the participating sources except for the non-transactional source, i. e. the WfMS. After all sources have sent their ready-to-commit, the FDBS starts the WfMS with its piece of work. Now, the result of the WfMS is decisive for the whole transaction. If it succeeds, a commit is sent to all the other sources. If the WfMS fails, the FDBS forwards an abort to the databases. Thus, we can guarantee the consistency of our data.

Comparing our approach with the requirements described in Sect. 2, we have proposed the following solutions. In the client interface, global functions are hidden behind SQL queries which are processed by the FDBS. Its query processor has to forward those parts of the global query to the WfMS which concern the predefined function access. The correct execution of the local functions is then realized by the workflow engine. In addition, it manages the heterogeneous interfaces to the integrated sources and keeps them transparent to other middleware components. Regarding distributed updates we propose to apply the concept of compensation spheres.

6. Related Work

Many approaches to support the integration of functions in addition to the integration of data choose object-oriented concepts [1, 4, 7, 17]. Such techniques enable the description of the structural characteristics of a source as well as the behavior of the instances by means of the definition of methods and functions. These approaches follow up the operational mapping apart from the structural mapping of the pure data integration. The operational mapping defines correspondences between operations on different levels. The operational integration then extends the application area of integration from the reuse of data to the reuse of data and application software. The referenced approaches do not provide a general methodology comparable to [16] for the schema integration. Instead, all the platform heterogeneities are solved in the proprietary implementations of the global functions. Furthermore, there are no or just a few means for the modeling of semantics in the global schema. However, more complete and declarative specifications may facilitate the process of integration and contribute to the understanding of the system and function dependencies.

Another approach is presented in [18]. The concept of megaprogramming considers the composition of components provided by heterogeneous, autonomous, and distributed software modules as methods, the so-called megamodules. The goal is to compose the methods in order to develop new applications and, at the same time, to keep the autonomy of the software modules. Megaprogramming focuses on the horizontal integration, i. e. combining components rather than realizing an integrated access to

the selected functionality of the integrated source systems whereas our approach strives for a vertical integration. In addition, the possible scenario, in which various schemas of the source systems may overlap, and the arising dependencies are not considered. Also, a query language is missing, since the modules are only used to develop new applications. Thus, a flexible interaction of the user is not possible.

7. Summary and Outlook

In this paper, we have presented an approach for the integration of heterogeneous data sources accessible via generic queries or predefined functions. The consideration of predefined functions has been motivated by current system environments where databases and applications are encapsulated providing an API with functions instead of a DB interface. After having described the requirements an appropriate solution should meet, we have introduced our integration architecture. It is based on two core components: an FDBS realizing the data integration and a WfMS implementing the predefined function access. The combination of both engines is realized by UDFs representing the results of function calls as relations within the FDBS. Moreover, we have divided our approach into a plan generation and a plan execution model. Focusing on the predefined function access, a description language has been developed for the function mapping. This language is based on XML and describes the mapping by means of dependencies between function parameters. Our intention is to keep the description language simple, lightweight, and, especially, independent from the implementation. Considering the plan execution model, we have presented solutions for the connection between the FDBS and the WfMS and the support for distributed transaction management.

At the moment, the presented mapping language is very rudimentary. Therefore, we are currently refining it, trying to exploit the whole functionality of XML as well as related standards and drafts. Furthermore, we will evaluate the performance of the WfMS. If the results meet our expectations we have to analyze the connecting alternatives between FDBS and WfMS in greater detail. If not, alternatives for the plan execution model will be developed.

References

[1] E. Bertino, M. Negri., and L. Sbattella. An Overview of the Comandos Integration System. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems*, pages 379–422. Prentice Hall, 1995.

[2] B. Czejdo, M. Rusinkiewicz, and D. W. Embley. An Approach to Schema Integration and Query Formulation in Federated Database Systems. In *Proc. 3rd Int. Conf. on Data Engineering (ICDE'87)*, pages 477–484, 1987.

[3] Enovia Corp. ENOVIAVPM. http://www.enovia.com/solutions/html/edesvpmoverview.htm, 2000.

[4] R. Gagliardi, M. Caneve, and G. Oldano. An Operational Approach to the Integration of Distributed Heterogeneous Environments. In *Databases: Theory, Design, and Applications*, pages 110–124, 1991. Postconference publication of PARBASE-90, 1st Int. Conf. on Databases, Parallel Architectures and their Applications.

[5] T. Härder and A. Reuter. Principles of Transaction Oriented Database Recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

[6] T. Härder, G. Sauter, and J. Thomas. The Intrinsic Problems of Structural Heterogeneity and an Approach to their Solution. *VLDB Journal*, 8(1):25–43, 1999.

[7] M. Härtig and K. R. Dittrich. An Object-Oriented Integration Framework for Building Heterogeneous Database Systems. In *Proc. IFIP DS-5 Conf. on Semantics in Interoperable Database Systems*, pages 33–53, Australia, 1992.

[8] K. Hergula. A Mapping Language for the Integration of Functions. Technical Report, DaimlerChrysler AG, 2000.

[9] ISO/IEC. Database Language SQL – Part 9: SQL/MED, November 1999. Final Committee Draft.

[10] F. Leymann. Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. In *Proc. of the German Conference BTW'95*, pages 51–70, Berlin, 1995.

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

[12] W. Meng and C. Yu. Query Processing in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 551–572. Edison Wesley, 1995.

[13] F. F. Rezende and K. Hergula. The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways. In *Proc. 24th Int. Conf. on Very Large Data Bases (VLDB'98)*, pages 146–157, New York, August 1998.

[14] SAP AG. SAP R/3. http://www.sap.com/solutions/r3/, 2000.

[15] SDRC Corporation. Metaphase. http://www.sdrc.com/nav/software-services/metaphase/, 2000.

[16] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[17] M. Tork Roth and P. M. Schwarz. Dont Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proc. 23th Int. Conf. on Very Large Data Bases (VLDB'97)*, pages 266–275, Athens, 1997.

[18] G. Wiederhold, P. Wegner, and S. Ceri. Towards Megaprogramming. *Communications on ACM*, 35(11):89–99, 1992.

[19] Workflow Management Coalition. Interface 1: Process Definition Interchange Process Model, October 1999. Version 1.1.

[20] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/REC-xml, 1998. W3C Recommendation.

[21] World Wide Web Consortium. XML Linking Language (XLink). http://www.w3.org/TR/xlink, 1999. W3C Working Draft.