Tightly-coupled Wrappers with Event Detection Subsystem for Heterogeneous Information Systems

Christopher Popfinger and Stefan Conrad
Databases and Information Systems
University of Düsseldorf, D-40225 Düsseldorf, Germany
{popfinger, conrad}@cs.uni-duesseldorf.de

Abstract

The interconnection of heterogeneous and autonomous data sources for information sharing demands for flexible and extensible integrative components. In this paper we present a universal architecture for building wrapper components to access various types of heterogeneous information sources. The wrapper comprises an event detection subsystem to detect and propagate modifications of the data stock. The architecture proposed especially supports Enhanced Active Database Systems*, which are able to actively notify their tightly-coupled wrapper components about data changes.*

1 Introduction

Wrappers are a well-known concept to the database community. They encapsulate heterogeneous data sources to provide clients with a common query interface. Queries and data are translated from the data model of the source to the data model the client expects and vice versa. Besides the query translation, a wrapper can be used to extend the functionality of an encapsulated data source. Additional functions are implemented in the wrapper and offered to the client or mediator.

Although many research projects that deal with information sharing among multiple heterogeneous sources rely on event detection to notify mediators of data or schema changes, there is only a few detailed information available. The TSIMMIS System [1], for example, uses template-based wrappers to access information from a variety of heterogeneous information sources. Although the system supports constraint management, there is no detailed information about the detection of events in the data stock. Another wrapper architecture for heterogeneous data sources is presented in [8] which contains a *schema change detector* to report schema changes to the mediator. Again, there is no description of the detection and notification process.

In this paper we present a universal wrapper architecture for heterogeneous information systems supporting various types of data sources. It comprises an event detection subsystem to react on modifications of the local data stock. The architecture is especially designed to wrap database systems with enhanced activity which are able to actively notify the wrapper component about data changes.

We start introducing Enhanced Active Database Systems in Section 2, followed by a description of the components of the architecture in Section 3. Section 4 explains the event detection and notification mechanisms of the event detection subsystem, while Section 5 concludes and draws up future work.

2 Enhanced Active Database Systems

Active database systems assist applications by migrating reactive behavior from the application to the DBMS. They are able to observe special requirements of applications and react in a convenient way if necessary to preserve data consistency and integrity. The integration of active behavior in relational database systems is not particularly new and most commercial database systems provide an active rule system. Unfortunately, the scope of triggers and stored procedures in active database systems has until recently been limited to the isolated databases they were defined at. Subsequent developments integrated special purpose programming languages (e.g. PL/SQL) into the database management system to overcome some limitations of the query language and to provide a more complex programming solution for critical applications. But again, the scope of these extensions was strictly limited to the system borders of the database system, so an interaction with its environment was very limited or simply impossible. Recent developments, especially in commercial database systems, take their functionality beyond former limits. The significant improvement, on which this work is based on, is the ability of modern database systems to execute programs or methods written in a standalone programming language from within triggers and stored procedures. In the following, the ability

of a database system to execute programs or methods from within its database management system to interact with its execution environment shall be called *enhanced activity*. A database system with enhanced activity is an *Enhanced Active Database System (EADBS)*. The execution of a program or method in this context shall be called an *External Program Call*.

The execution of complex programs from inside a DBMS offers new perspectives to data management in a database system. It can, for instance, be used to perform complex interactions with its execution environment, like global integrity maintenance in federations. In this paper, we use the enhanced activity to enable a database system to interact with its corresponding wrapper component. It establishes a connection and signals changes of the data stock to the wrapper's event detection subsystem. Due to this bidirectional communication of the wrapper and the underlaying database system, we say the wrapper is *tightly-coupled* to the data source. The programming language for coding the external programs must be able to interact with other standalone programs via sockets or a Remote Method Execution Protocol. Within recent commercial database systems a commonly supported programming language which meets these requirements is Java (e.g. Java Stored Procedures or Java UDFs). It comprises a comprehensive package list for networking and remote method invocation. Furthermore, it contains JDBC, a common database connectivity framework, to provide a standardized interface for a multitude of different data sources like relational databases or even flat files (CSV). Although we exemplify our concept in the context of Java, it can be adapted to other database systems supporting different programming languages that fulfill the requirements just mentioned.

3 Wrapper Architecture

We now present the wrapper architecture for various data sources, which comprises both, a query subsystem and an event detection subsystem to detect and propagate modifications of the data stock. Our architecture is especially designed for the implementation of heterogeneous information systems where data is stored on multiple autonomous sources that are interconnected by a mediation layer. It particularly provides a notification interface for Enhanced Active Component Systems to actively signal modifications of the data stock to the tightly-coupled wrapper (see 4.2). The components of the wrapper itself communicate via methods and function calls written in the wrapper programming language and return data types or objects. Figure 1 depicts the components of the wrapper architecture, which will be described in the following.

Heterogeneous Data Sources Our architecture basically supports any type of data source, but the functionality that can be offered to the mediation layer by the wrapper strongly depends on the capabilities of the underlaying data source. A data source is connected to the wrapper by a source specific query interface. Types of data sources can be, for example, unstructured or semi-structured files, relational or object-oriented database systems, or directory systems, with our without integrated active mechanisms. The architecture is especially designed to wrap EADBSs with the ability to signal data modifications to the wrapper.

Source Specific Query Interface The source specific query interface is a software component that provides basic operations on the data stock. It knows how to open and close a connection to a source and what types of queries are supported. Furthermore it translates query results into a representation format of the programming language. This can be, for example, a result set, a multidimensional array, or a heterogeneous collection of basic data types. The required query language and the returned query results significantly depend on the functionality provided by the data source. Most databases with database management systems support SQL as a standardized query language and return a resource identifier which points to the result of a specific query. A well-established package which can be associated to this architectural component is the Java Database Connectivity Framework (JDBC). It supports various types of data sources including even query interfaces for flat or XML files. The query manager calls functions of the source specific query interface to communicate with the data source and to read or modify its data stock.

Query Manager The query manager consists of a set of functions which encapsulate the entire communication from the wrapper components to the data source. These functions are specifically designed for the desired data source functionality of the overall system. The query manager uses the source specific query interface to open and close connections, send queries, and receive corresponding results. Each query function executes a single query or a parameterized family of queries to read or modify the data stock. The functions can be divided into two types of interaction: data requests and repository requests. Data request functions provide an interface to access local data in the data source, whereas repository requests are used to access meta data stored in the repository. Both, controller and event monitor, call the predefined query functions to interact with the data source. They receive the corresponding query result as data types or objects as return values from the functions. Due to the centralization of the data source access in the query manager and the separation of query manager and source specific query interface we achieve a portable wrapper solution for heterogeneous information systems.

Converter Since a wrapper is used to convert queries and data from one model to another, we need a component to convert source specific data into an exchange format the

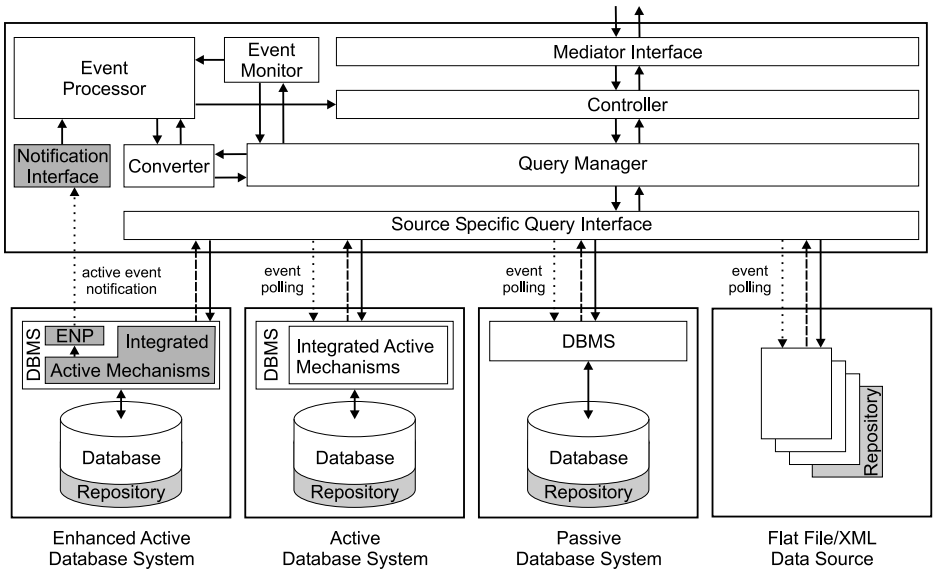

Figure 1. Wrapper architecture with event detection subsystem for various types of data sources.

mediation layer expects and vice versa. This conversion is done by the converter component. On the one hand, it is used by the query manager and the event processor to transform local data into the desired exchange format before it is send to the mediation layer. On the other hand it converts external data received from the mediation layer into the internal representation format understandable to the query manager. In this paper we do not provide a specific exchange format, since the choice of a format depends on the properties, characteristics, and functionality of the overall system. However, a common exchange format for the use in the mediation layer of a heterogeneous information systems could be based on RDF or OWL.

Controller The controller controls the interaction between the mediation layer and the data source. In particular, it maps external requests to internal query functions of the query manager, and forwards results for output. An external request may result in a sequence of database operations which are also coordinated and processed by the controller component. Furthermore, it manages all the meta data required in the repository. The controller also provides a set of event notification functions, which are used by the event processor to signal data modifications of the local source. Events can thus be reported to the mediation layer for further processing.

Mediator Interface The mediator interface is the communication unit for interaction with the mediation layer. The functionality and concrete implementation of the interface depends on the type of infrastructure to be established. Possible infrastructures for mediation layers could be based on for example RMI or CORBA, Server-Client communication over specified protocols, web services, publish-subscribe, or peer-to-peer. The mediator interface establishes connections to remote systems and handles incoming or outgoing requests between the mediation layer and the wrapper controller.

Repository The repository contains all meta data required for the operation of the wrapper component and the interaction with the mediation layer and the data source. This includes configurations and properties, data source descriptors, login information, and access control lists. The repository is managed exclusively by the controller of the wrapper, but it is stored directly in the local data source. Thus, requests to data and meta data in the repository can be processed in a uniform manner by the query manager. The query manager provides the controller and event monitor with query functions for repository management and access. The main advantage is the homogeneous storage of both, data and meta data, and the centralized access via the query manager.

Event Monitor The event monitor is the active component of the event detection subsystem. It detects data modifications in data sources that are not able to actively propagate events to the wrapper, like for example active database systems without enhanced activity. The event monitor obtains information about the data items to be scanned from the repository. Using this information it periodically scans relevant parts of the data stock for modifications. In the case of a relational database a *monitoring schedule* in the repository could contain the name of a relation and the time period in which the relation should be scanned. The overall meta data required for monitoring data modifications de-

pends on the change detection algorithm implemented by the event monitor, which in turn depends on the type of local data source. The event monitor is executed as a subprocess (thread) at startup of the wrapper and does not provide any operational interface to the remaining wrapper components. Like the controller, the event monitor uses the query functions of the query manager to access the data source. If a relevant data modification is detected, the modified data items are extracted from the data source and reported to the event processor.

Notification Interface The notification interface is the passive component of the event detection subsystem of our wrapper architecture. It receives all update notifications directly from an EADBS via an external notification program (see 4.2 for details). It consists of a source specific set of functions, which are called by the underlaying EADBS. After a change has been signaled to the notification interface, the modified data items are reported to the event processor.

Event Processor All events, monitored by the event monitor or signaled by the notification interface, are processed by the event processor. It receives modified data items and converts them into the exchange format using the converter component. Afterwards, it forwards the changes directly to the wrapper controller.

4 Event Detection

The architecture presented in this paper comprises components for the detection and notification of events, i.e. mainly modifications of the data stock. The detection and propagation of data modifications is the basis for many research projects (e.g. [1, 2]) which deal with the interconnection of heterogeneous and autonomous data sources to share information. A common characteristic of these approaches is the use of a notification mechanism which somehow notifies a mediating component (e.g. a constraint manager) of an event occurring in the local data source to trigger a certain subsequent operation. Such notification mechanisms are often based on the active capabilities of an underlaying database management system, but there is so far no detailed description of this interaction available.

The event detection subsystem of our architecture consists of two main parts: event monitoring, for data sources without enhanced activity, and active event notification. They are described in the following sections.

4.1 Event Monitoring

Up to now, there are many types of data source that do not have the enhanced activity which is required to actively notify the wrapper of data modifications. Usually, flat or XML files are unable to monitor transactions or to maintain integrity constraints, so for these sources it is impossible to detect changes in their own data themselves. This also applies to database systems: as long as the database management system do not provide the enhanced activity, it are unable to interact with its environment, even if triggers are provided. Thus, the active event notification to the wrapper component is simply impossible with these types of data sources. To detect events in these data sources, we have included an event monitor into our architecture. It accesses the data source via a specific set of query functions provided by the query manager. The information about the entities and attributes to be scanned are created by the controller and stored as *monitoring schedules* in the repository. A monitoring schedule contains the entities and attributes which are monitored and the time period in which the scan has to be repeated. Additional information depends on the specific algorithm of the monitoring process.

There are several approaches for change detection in different kinds of data sources that can be applied to this part of the subsystem. Changes to flat files can be detected using string comparison algorithms, like presented in [4]. Solutions for relational data and hierarchically structured data, such as nested-objects, are presented in [5] and [3] respectively. Due to the popularity of XML there is a number of algorithms which deal with the comparison and change detection in XML files, e.g. [9]. After changes have been identified by the event monitor, it propagates the modified data items to the notification interface, where they are transformed into an internal representation format and sent to the controller. The processing of the updates depends on the implementation. The updates could be used to check global integrity constraints, to accumulate changes for future synchronization as needed for mobile databases, or to actively replicate data items to corresponding data sources via the mediation layer.

4.2 Active Event Notification

The active propagation of events occurring in the data stock can be realized using EADBSs. As mentioned above this type of database systems is able to interact with its environment using external program calls from within triggers. The external programs are generally written in a standalone programming language and stored as functions inside the database system. Since Java is supported in most commercial database systems with enhanced activity, we exemplify the active event notification process using concepts of object-oriented programming languages. The concept proposed certainly adapts to other languages and data models.

To detect data modifications we create triggers on the entities and attributes we want to monitor. The triggers can easily be adjusted to react on insertions, deletions, or updates on the data stock. After such an event has been detected, the trigger calls an *external notification program* (ENP), which is responsible for the notification of the wrapper. As parameters it passes the data items affected and cor-

responding meta data (e.g. the type of modification). The ENP is executed by the database system and connects to the notification interface of the wrapper, whereupon it transmits the changed data items. The notification interface in turn reports to the event processor.

The main challenge in this process is the communication between the external program of the data source and the notification interface of the wrapper. During the implementation we face the problem that the wrapper component and the ENP are two different processes which are initiated and executed independently from each other. A wrapper component is installed and started once for each data source as a standalone application. An ENP is initialized and executed by the database system whenever a corresponding trigger is activated by a database transaction. An ENP can be used by one or more triggers, but we are also able to implement a single ENP for each trigger. After the notification, the external program process terminates and resources are freed again. In fact, we have one instance of the notification interface and many instances of one or more ENPs.

In our architecture the interaction of ENPs and the wrapper is based on the *event notification pattern* [7], which is closely related to the *observer pattern* of Gamma et al. The wrapper controller (observer) wants to be informed about specific state changes of the database (subject) as soon as possible. A change of state is encapsulated in an instance of an ENP. The ENPs forward the event to the notification interface, which acts like an observer's event stub or event listener. The event stub calls the required operations of the observer to react on the event. All events, detected by the event monitor or notification interface, are both signaled to the controller via the event processor. The event monitor encapsulates changes of the database state and reports to the event processor in the same way as the ENPs.

As already mentioned, the notification interface and the notifiers (excluding the event monitor) are separate executables and they do not necessarily have to reside on the same node of the network. In the case of Java as an interpreted language, they do not even have to be executed by the same Java Virtual Machine, since recent database systems usually ship with their own Java interpreter. Thus, the communication between the ENPs and the notification interface has to be established via sockets, a distributed object protocol such as CORBA, or a Remote Method Execution protocol such as Java RMI. In the case of Java we suggest Java RMI for communication between the components. Please note that an EADBS can also be monitored by the event monitor like passive data sources, if the creation of triggers would impose unacceptable restrictions to local autonomy.

5 Conclusion and Future Work

In this paper we have presented a universal wrapper architecture for the use in heterogeneous information systems with autonomous component systems. Besides the query interface it comprises an event detection subsystem to detect modifications of data in the underlaying source. The architecture especially supports *Enhanced Active Database Systems*, which are able to actively notify their corresponding wrapper about events occurring in the database. The communication between EADBSs and the wrapper is realized using a Remote Method Execution Protocol implementing the event notification design pattern.

In the next steps we finish the implementation of prototypes for a set of different relational databases and evaluate the wrapper architecture in the context of the DÍGAME architecture [6] for a real world scenario. Concurrently we are developing a framework for the implementation and deployment of event notification programs for diverse EADBSs, which shall later be included into a toolbox to assist the creation and setup of wrapper components. We furthermore plan to create prototypes for other types of data sources (e.g. XML) and evaluate different change detection algorithms for event monitoring.

References

[1] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, 1994.

[2] S. Chawathe, H. Garcia-Molina, and J. Widom. A Toolkit For Constraint Management In Heterogeneous Information Systems. In *Proc. of the Int. Conf. on Data Engineering*, pages 56–65, 1996.

[3] S. S. Chawathe and H. Garcia-Molina. Meaningful Change Detection In Structured Data. In *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 26–37. ACM Press, 1997.

[4] J. W. Hunt and T. G. Szymanski. A fast Algorithm for Computing Longest Common Subsequences. *Commun. ACM*, 20(5):350–353, 1977.

[5] W. Labio and H. Garcia-Molina. Efficient Snapshot Differential Algorithms for Data Warehousing. In *Proc. of the 22th Int. Conf. on Very Large Data Bases*, pages 63–74. Morgan Kaufmann Publishers Inc., 1996.

[6] C. Pérez de Laborda, C. Popfinger, and S. Conrad. Dynamic Intra- and Inter-Enterprise Collaboration Using an Enhanced Multidatabase Architecture. In *16th Int. Conf. and Workshop on Database and Expert Systems Applications*. IEEE Computer Society Press, 2005.

[7] D. Riehle. The Event Notification Pattern - Integrating Implicit Invocation with Object-Orientation. *Theory and Practice of Object Systems*, 2(1):43–52, 1996.

[8] H. Wang, J. Li, and Z. He. An Effective Wrapper Architecture to Heterogeneous Data Source. In *AINA*, pages 565–569, 2003.

[9] Y. Wang, D. J. DeWitt, and J. Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *ICDE*, pages 519–530, 2003.