Dynamic Intra- and Inter-Enterprise Collaboration Using an Enhanced Multidatabase Architecture

Cristian Pérez de Laborda Christopher Popfinger
Stefan Conrad
Databases and Information Systems
Heinrich-Heine-Universität Düsseldorf, D-40225 Düsseldorf, Germany
{perezdel, popfinger, conrad}@cs.uni-duesseldorf.de

Abstract

Modern intra- and inter-enterprise collaboration requires access to information spread over multiple autonomous and heterogeneous data sources. In this paper we present a loosely coupled multidatabase architecture enhanced with P2P concepts. It achieves a reasonable tradeoff between autonomy and information sharing among both, permanently available and volatile data sources. Each data node decides autonomously which kind of information to share. Data availability, query performance, and up-to-dateness on each participating data node is improved using a push-based replication strategy, which propagates data modifications over multiple nodes.

1 Motivation

Since the first centralized databases found their way into the enterprises in the late 60s, the needs and requirements have changed towards a more distributed management of data. Today there are many corporations which possess a large amount of databases, often spread over different regions or countries and generally connected to a network. These local databases typically raised in an autonomous and independent manner fitting the special needs of the users at the local site. This leads to logical and physical differences in the databases concerning data formats, concurrency control, the data manipulation language, or the data model [8]. It is crucial for a company to keep track of its distributed data in such a heterogeneous environment. Cooperating departments need shared access to this data, to be able to increase their productivity. Multidatabases were introduced for this reason, in order to integrate data from heterogeneous sources.

One of the main challenges in the integration of data in such environments is the autonomy of the participating data nodes. This autonomy implies the ability to choose its own database design and operational behavior. Local autonomy is tightly attached to the data ownership, i.e. who is responsible for the correctness, availability, and consistency of the shared data. Centralizing data means to limit local autonomy and revoke the responsibility from the local administrator, which is not reasonable in many cases. A federated architecture for decentralizing data has to balance both, the highest possible local autonomy and a reasonable degree of information sharing [6]. Hence, the architecture of a company wide information system has to be applicable to the data policy of the company and vice versa. To be more precise, the question of data ownership determines the composition of the company wide information platform, while it has to ensure a high level of consistency and fail-safety.

In this paper we describe the DÍGAME architecture, a **D**ynamic **I**nformation **G**rid in an **A**ctive **M**ultidatabase **E**nvironment, which connects heterogeneous and autonomous data sources to support loosely coupled intra- and inter-enterprise collaboration. We have enhanced the multidatabase architecture of Heimbigner and McLeod [6] with Peer-to-Peer (P2P) concepts to offer a flexible information grid with high data availability to provide each participating node of this grid with all the data required. Extending the approach of Heimbigner and McLeod, our architecture enables the sharing of information among both, permanently available and volatile data sources without any central component. For that we include a push-based replication mechanism which propagates data modifications over multiple nodes. Thus, we are able to ensure high availability of data, even if the original data source is temporarily unreachable. Additionally, the replication of data increases query performance, since we do not have to query remote data sources. An information sharing environment, which comprises the information shared by interconnecting heterogeneous and autonomous data peers using our architecture, shall in the following be referred to as an information or data grid [3].

The remainder of this paper is organized as follows. In section 2 we start describing the architecture of our dynamic information grid with an overview of the basic functionality. Afterwards we discuss the major characteristics in section 3, followed by a description of our first wrapper prototype in section 4. Section 5 discusses related work and section 6 concludes and draws up future work.

2 DÍGAME Architecture

2.1 Basic Functionality

We now draw up the basic functionality of our enhanced multidatabase architecture using a motivating example.

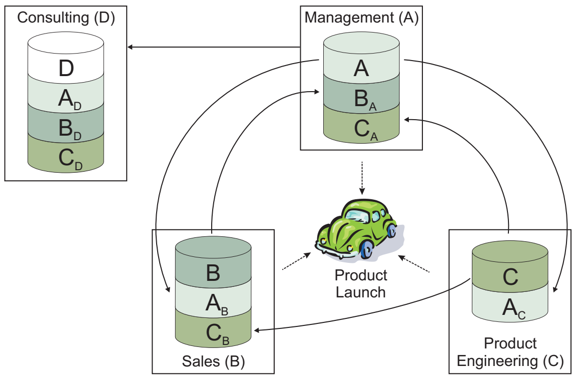

Figure 1. Collaborative Work with DÍGAME

Consider a worldwide operating company planning the launch of a new product (Fig. 1). We assume that there are three departments involved in this business process: the executive board (management), the sales office, and the product engineering department. Each department manages its own database to store the information for which it is responsible in an autonomous way. The management produces basic data of the product (A) including deadlines, descriptions, workflows, and additional objectives. This management information is substantial for the further product development and the work in the participating departments. The product engineering department uses a predefined part of that management data (A_C) as basic conditions for the concrete implementation and technical realization of the product. Local applications create additional data which has to be stored separately (C). According to the product engineering the sales department enriches the authoritative management data (A_B) with concrete concepts for the upcoming product launch (B). Furthermore concrete development plans of the product engineering are required to prepare sales strategies (C_B). Both, sales and product engineering departments, concretize the strategic guidelines of the management in their specific assignment. To keep track of the costs and the progress of the project, it is indispensable for the management to access the product engineering and sales department's relevant information just mentioned (B_A, C_A).

Sharing data within this company using our architecture is realized using a push-based replication strategy to improve data availability, query performance, and up-to-dateness on each participating data node. Hence, the data source actively propagates data updates to relevant peers, which are herewith able to maintain an up-to-date replica of the imported data. For example, the creation of a new replica of management data on department B is realized as follows: each data source of the departments A, B and C is wrapped by a source-specific wrapper component. These wrappers build up a communication layer, which enables the departments to interact pair-wise using a common protocol. This union adopts Peer-to-Peer concepts and operates without any central administrative instance. Due to these characteristics the combination of such a data source and its related wrapper component can be named as a (data) peer.

The administrator of peer A makes a subset of its own local data accessible using the administrative interface of the wrapper component. The export schema [6] created this way is managed by the wrapper component and specifies the information that the department is willing to share. The information concerning the access control to local data by remote peers is attached to the export schema. Peer B is now able to import the data into its local database subscribing to a specific part (A_B) of the published data, i.e. the data required by the department. During this subscription process the data target (subscriber) informs the data source (publisher), which subset of the export schema it is willing to import. The data stock A_B is then transferred to the subscriber to perform an initial filling.

If a data or schema modification is detected by the wrapper of the publisher, all relevant subscribers have to be informed. To determine whether the subscribers, including peer B, have to be notified about this modification, the wrapper queries all export schemas in the repository. The modified data or schema items are then pushed actively to the relevant subscribers using a semantically rich representation format. Each data peer is herewith able to maintain an up-to-date replica of the data and schema items required by local applications.

Now the management department has decided to involve an external consulting group D to analyze and optimize the productivity within corporate workflows. Therefore the consultants need access to the entire management data, including the data of departments B and C. Instead of negotiating separate data exchanges with every single department, our architecture enables the consulting group to obtain all data required from only one data source, the management department. This can be realized, since our architecture supports the sharing of data imported from other nodes. Please note, that the export of imported data must explicitly be allowed by the administrator of the management department. After the consulting has subscribed to the entire management data, data updates in B and C are propagated to A as usual. Node A delivers updates on its own data stock and additionally those coming from nodes B and C to its subscriber D. Due to this characteristic, node A becomes a Data Hub for the consulting group according to the Link Pattern Catalog introduced in [12].

We distinguish two different types of update propagation: *direct* and *indirect updates*. After an update is detected on local data of a data source, it is propagated to the relevant subscribers. Referring to our example, B gets direct updates, whenever modifications occur on the data stock of node C. If a node explicitly shares a previously imported data stock, its modifications are in turn propagated to other subscribers. Referring to our example, node A shares previously imported data from peers B and C, which is subscribed by the consulting group node D. If an update occurs on B or C, it is first propagated to node A, which in turn propagates it to its subscriber D. This sequence of update propagation is called a *cascading update*.

Further partners may join this collaboration at any time. In fact, each peer can be provided with any data concerning the product launch stored in one of the collaborating data nodes without interfering with existing data flows. The data source maintained by the partner can on the other hand be easily connected to the existing data grid sharing its own data. If a peer is no longer willing to share its data, it can easily be removed from the data grid, notifying all its subscribers to remove the replicas from their local data stocks. The support of this temporary collaboration makes our DÍGAME architecture particularly suitable for virtual cooperations.

2.2 Components of the Architecture

In this section we discuss the components of our DÍGAME architecture (Fig. 2). The data grid $DG := (P, C)$ created by our architecture is a directed graph, which consists of a set of peers $P := \{p_1, ...p_n\}$ and a set of connections C, where a connection $c = (p_i, p_j) \in C$ links exactly two peers, representing a data flow from p_i to p_j.

As already mentioned, each peer consists of a component database and a corresponding wrapper component. These components which are both required for establishing data flows between communicating peers are described in the following:

Wrapper: The core of our data grid is the wrapper component, which provides a uniform interface to the heterogeneous component databases. It is responsible for negotiating and establishing communication among peers and coordinates the data and schema exchanges after a communication channel has been set up. Each wrapper maintains a repository in its corresponding data source to store information about subscribers, export and import schemas, access control lists, and delivery schedules.

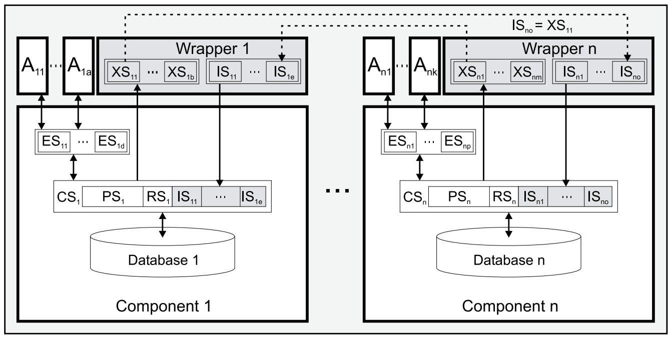

Figure 2. DÍGAME **Architecture**

Each wrapper has to realize two major tasks: exporting and importing data and schema items. To export local data from a peer p, a set of export schemas $\mathcal{XS}_p := \{XS_{p1}, ..., XS_{pi}\}$ is maintained by the wrapper of p. To allow indirect updates, those export schemas have to be based on the entire conceptual schema CS_p of the database, excluding RS_p, the schema of the repository stored on p, i.e. $\forall XS \in \mathcal{XS}_p : XS \subseteq CS_p \setminus RS_p$. They are required to determine, which peers have to be informed about data modifications. Since exporting peers actively propagate the data and schema to relevant subscribers, they must be able to detect modifications on their local data stock. Earlier research proposes several mechanisms helping a wrapper to monitor data modifications [13]. If there are triggers of underlaying database systems available, they should be used, particularly their enhanced functionality given by recent developments in database systems [11].

To import data from a remote peer p, the wrapper on a peer q ($p \neq q$) maintains a set of import schemas $\mathcal{IS}_q := \{IS_{q1}, ..., IS_{qj}\}$, where

$$\forall q \in P\, \forall IS \in \mathcal{IS}_q\, \exists_1 p \in P\, \exists_1 XS \in \mathcal{XS}_p : IS = XS \wedge p \neq q \quad (1)$$

and

$$\forall p \in P\, \forall XS \in \mathcal{XS}_p\, \exists_1 q \in P\, \exists_1 IS \in \mathcal{IS}_q : XS = IS \wedge q \neq p \quad (2)$$

After the initial import of subscribed data, each data and schema modification propagated by remote peers is reproduced locally in the workspace of the wrapper.

Autonomous Component Databases: According to the Three Schemas Architecture and the architecture for loosely coupled multidatabases [6], each component database on a peer q contains several types of schemas (see Fig. 2). The private schema PS_q stores data, which is locally produced and maintained. It is controlled exclusively by the local database administrator. Other peers do not have direct access to this data. Besides the private schema, the conceptual schema CS_q comprises the disjoint union of the import schemas and the repository mentioned above, i.e. $CS_q := (\dot{\bigcup}_{IS \in \mathcal{IS}_q} IS) \cup PS_q \cup RS_q$, where $IS \cap PS_q = \emptyset$.

Local applications $A_{q1}, ..., A_{qf}$ can now access and process the data of the conceptual schema excluding the repository information as usual using a set of external schemas $\mathcal{ES}_q := \{ES_{q1}, ..., ES_{qd}\}$. The only limitation is the read-only access to data derived from the imported schema.

Please keep in mind that the imported data and the repository are exclusively managed by the wrapper component and should never be modified by the local administrator or applications, although this would be possible due to the local autonomy. In fact, future implementations could support such multi-master replication techniques.

3 Characteristics

In this section we discuss the major characteristics of our DÍGAME architecture including the advantages and limitations related to its implementation.

Autonomy and Heterogeneity: Our architecture is based on the concept of loosely coupled multidatabases of Heimbigner and McLeod [6] using import and export schemas for data exchanges. The aim of this architecture is to achieve a feasible trade-off between local autonomy and a reasonable degree of information sharing. A data source is basically free to decide on its own level and form of participation. This includes the ability to decide which data it is willing to export, which data is imported, and during which periods services are provided.

Our architecture supports the integration of principally any kind of data source using a wrapper component tailored to that specific data source. The wrapper provides an uniform interface for the DÍGAME system, where communication is performed using a standardized protocol and exchange format.

No Central Authority: Any information sharing environment based on our DÍGAME architecture interconnects autonomous and previously isolated data peers. Each participating data node keeps full control over its own data, i.e. there is no central authority imposing certain restrictions. Contrary to other approaches like [14] we do not use any central component, where publications or subscriptions are managed. In our system, peers subscribe directly to data published by other nodes. The information on the data offered is not managed centrally, but stored exclusively on the corresponding peers.

Wrapper organized similar to P2P systems: We have enhanced the multidatabase architecture with P2P concepts. The wrappers in our architecture interact similar to classical P2P networks. Data exports and imports are exclusively negotiated pair-wise, whereas each peer is basically able to interact with any number of data nodes. The entire communication is realized without any central authority, resulting in a network of self-responsible peers, where members are basically able to join or leave at any time.

Replication: The replication of data is one of the main features of our DÍGAME architecture. Data availability is improved in the information grid allowing a data stock to be directly or indirectly replicated over multiple peers. This means, that required data is accessible, even if the original data node is temporary unavailable. Furthermore query performance is increased, since all the required data is stored locally.

The refreshment strategies for updating the replicas depend on the application field. We are basically not limited to a single delivery schedule, but able to provide specific replication strategies depending on the needs of each subscribing peer. Generally, the preferred delivery schedule is an immediate propagation of updates, but other possible delivery schedules can be, but are not limited to periodical or even aggregated propagation. The replication is managed by the wrapper component, which holds information about each subscribing database and its corresponding delivery schedule in its corresponding repository. DÍGAME uses *lazy replication* protocols with one single master and multiple read-only replicas.

Push-based Protocol: A further central characteristic of our architecture is the push-based propagation of data and schema modifications to subscribing peers. At first a data peer subscribes to data offered by a data source, whereupon it receives once a complete copy of the requested data. Afterwards the data source pushes all relevant updates directly to the subscribers according to their specific delivery schedule. This modifications are passed on to further subscribers using indirect updates, until all replicas are updated. Each peer maintaining a replica of remote data is herewith able to access data, which is as up-to-date as possible, even if the original data source is temporary not available.

If a replica can not be updated, because a subscriber is currently not reachable, we have decided to include a pull-based fallback mechanism into our architecture. After the communication has been reestablished, the data target can then actively query the data source whether data updates have occurred since their last contact. Thereupon lost updates are propagated once again to the data target.

Standardized Exchange Format: The dynamic interconnectivity of data peers requires a standardized exchange format, suitable for both, data and schema representation. Using knowledge representation techniques we can guarantee that every single data peer understands data and schema updates without explicitly arranging an exchange format. The additional integration of identifiers for data items (e.g. [9]) within the data exchange process simplifies data maintenance, especially if data is imported from multiple sources. This meta information may furthermore be useful for detecting and solving conflicts within the data.

We have decided to use the Web Ontology Language OWL as the common representation format for our architecture, since it provides several advantages over classical (semi)structured exchange formats. Based on a meta representation of (relational) databases we can describe the schema of virtually any database. Thereupon the schema representation itself can be used as an OWL ontology, to base the representation of the actual data on. This flexible and powerful technique is only possible due to the possibilities given by *OWL Full* to interpret an instance of a metamodel as a novel ontology.

The representation of relational data and schema with the Web Ontology Language OWL entails several advantages over classical (semi)structured exchange formats like XML. A detailed discussion on this topic can be found in [10].

Local Integration: As already mentioned above, each peer may subscribe to multiple data sources. For each subscription it obtains an exact copy of the relevant remote data and schema items. Since we do not have a global schema, the imported data is integrated individually following local integration strategies, which are not provided by our DÍGAME architecture. Having all required data stored in the local database, we are particularly able to associate local and *remote* data with integrity constraints provided by the database, e.g. foreign keys. Furthermore index structures can be created on imported data to optimize data access according to local query requirements.

4 DÍGAME System Design

Now we give a brief overview of the design of our first DÍGAME wrapper prototype. The components required are depicted in Fig. 3. Our system is divided into two conceptual layers: a source specific and a source independent layer. Both layers interact exclusively using Java objects. The source specific layer contains all classes and packages, which are customized to the specific data source. This layer has to be implemented for each type of data source and provides a uniform interface for the classes of the source independent layer. Thus, new data sources can be supported exchanging solely the source specific components, leaving the remaining components unchanged.

The main component of our system is the DÍGAME Manager. It initializes the remaining components and coordinates the entire system flow. At first, it is responsible for the creation and management of import and export schemas, which are set up by the administrator using a corresponding user interface and stored in the repository of the wrapper. Thereupon, the DÍGAME Manager automatically handles subscription and unsubscription requests to shared data from remote peers and coordinates the data exchange during data import and export processes. After a data modification has been reported, it determines the peers which have to be provided with that updated data and initiates the corresponding data transfers. Outgoing data is compressed by the DÍGAME Manager using the Compression Unit to reduce network traffic.

The entire communication with the data source is handled by the Data Handler. It consists of a set of functions providing a uniform interface to the data stored on that specific source. This interface is used by the DÍGAME Manager and the event detection packages to access the data and the repository, which are stored both directly in the data source. Thus, the access to both, the storage of data and meta data are entirely managed by the Data Handler component. As a result, all source specific properties, i.e. data model, query language, or operating system are hidden from the components of the source independent layer.

The Data Handler supports three types of interactions: *data requests*, *repository requests*, and *event notifications*. The DÍGAME Manager component submits *data requests*, whenever a new peer subscribes to the data or data updates are received from remote peers. In the first case, data items have to be transformed into the OWL exchange format, in the latter from OWL into the source specific data format. These conversions are realized by the OWL converter, closely attached to the Data Handler.

Changes on the local data stock are signalized to the Data Handler via *event notifications* by the event processor. These events are either detected by the notification interface or the event monitor. If a data source supports extended triggers, the notification interface is directly invoked by database triggers to notify the event processor about local data modifications [11]. Otherwise, such events have to be detected by the event monitor, which therefore periodically scans the local data stock.

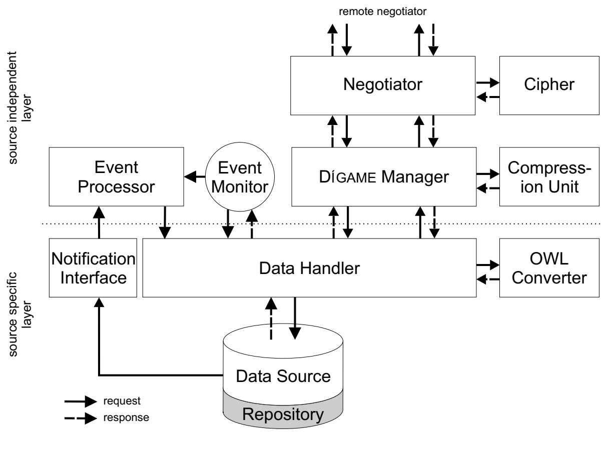

Figure 3. Design of the DÍGAME wrapper

A communication between two peers is established using the Negotiator. It interacts with remote peers using a special protocol to establish a secure communication channel. After the successful authentication and authorization it is used by the DÍGAME Manager for interacting with remote peers. The communication is encrypted using the Cipher component of our system.

5 Related Work

With the raise of filesharing systems like Napster or Gnutella [2] the database community started to seriously adopt the idea of P2P systems to the formerly known loosely coupled database systems. Contrary to the data grid, P2P database systems do not have a global control in form of a global registry, global services, or a global resource management, but multiple databases with overlapping and inconsistent data. These P2P databases resemble heterogeneous and distributed databases, also known as multidatabases [4]. Currently the database community makes a great effort in investigating P2P databases. Particularly the *Piazza* [5] project is worth mentioning, where a P2P system is built up with the techniques of the *Semantic Web* with local point–to–point data translations, rather than mapping to common mediated schemas or ontologies. Contrary to Halevy et al., we deal mainly with relational data and do not have a global schema, as every peer may have its own import–/export–schema combination. For a more general glimpse on data mappings in P2P systems see [7].

Our strategy allows data to be exchanged among distributed databases connected through a lazy network. This means, that although a running network may not be guaranteed and thus some data broadcasts may be lost, the system is able to heal itself. In contrast to the broadcast disks [1], we ensure that data is only broadcasted to the clients when changes occur, unless the communication between both peers crashes. Hence our approach resembles a *push–based* system with a *pull–based* fallback, similar to [1] with the major difference that our approach is not based on broadcast disks, but on a push–based replication strategy.

6 Conclusion and Future Work

In this paper we presented the DÍGAME architecture, which connects heterogeneous and autonomous data sources creating a dynamic information grid. This architecture enhances the well-known multidatabase architecture with P2P concepts, in order to support dynamic intra- and inter-enterprise collaboration. Local administrators decide themselves on their level of participation. Data provided by other peers can be subscribed and integrated into the local database as needed. The data source actively propagates changes on the subscribed data and schema items to the relevant peers via a standardized exchange format resulting in a replication of the data. Peers participating in the data grid interact pairwise without being managed by any central authority.

Further steps include the refinement of the prototype. In addition, we have to examine the impact of DÍGAME on the network traffic, particularly concerning query intensive applications.

References

[1] S. Acharya, M. Franklin, and S. Zdonik. Balancing Push and Pull for Data Broadcast. In *Proc. of the 1997 ACM SIGMOD Int. Conf. on Management of Data*, pages 183–194. ACM Press, 1997.

[2] B. Carlsson and R. Gustavsson. The Rise and Fall of Napster - An Evolutionary Approach. In *AMT 2001, Proc. of the 6th Int. Conference - Active Media Technology*, volume 2252 of *LNCS*, pages 347–354. Springer, 2001.

[3] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23(3):187–200, 2000.

[4] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What Can Databases Do for Peer-to-Peer? In *Proc. of the 4th Int. Workshop on the Web and Databases (WebDB)*, 2001.

[5] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proc. of the Twelfth Int. Conf. on World Wide Web*, pages 556–567, 2003.

[6] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Information Systems (TOIS)*, 3(3):253–278, 1985.

[7] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data*, pages 325–336. ACM Press, 2003.

[8] W. Litwin and A. Abdellatif. Multidatabase Interoperability. *Computer*, 19(12):10–18, 1986.

[9] C. Pérez de Laborda and S. Conrad. A Semantic Web based Identification Mechanism for Databases. In *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases (KRDB 2003)*, volume 79 of *CEUR*, pages 123–130. Technical University of Aachen (RWTH), 2003.

[10] C. Pérez de Laborda and S. Conrad. Relational.OWL - A Data and Schema Representation Format Based on OWL. In *Second Asia-Pacific Conf. on Conceptual Modelling (APCCM2005)*, volume 43 of *CRPIT*, pages 89–96. ACS, 2005.

[11] C. Popfinger and S. Conrad. Tightly-coupled Wrappers with Event Detection Subsystem for Heterogeneous Information Systems. In *DEXA Workshop Proc. of the 8th Intl. Workshop on Network-Based Information Systems (NBiS 2005)*. IEEE Computer Society Press, 2005.

[12] C. Popfinger, C. Pérez de Laborda, and S. Conrad. Link Patterns for Modeling Information Grids and P2P Networks. In *Conceptual Modeling - ER 2004, 23rd Int. Conf. on Conceptual Modeling*, volume 3288 of *LNCS*, pages 388–401. Springer, 2004.

[13] C. Türker and S. Conrad. Towards Maintaining Integrity of Federated Databases. In *Data Management Systems, Proc. of the 3rd Int. Workshop on Information Technology, BIWIT'97*, pages 93–100. IEEE Computer Society Press, 1997.

[14] J. Yang, M. P. Papazoglou, and B. J. Krämer. A Publish/Subscribe Scheme for Peer-to-Peer Database Networks. In *OTM to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *LNCS*, pages 244–262. Springer, 2003.