# Link Patterns for Modeling Information Grids and P2P Networks

Christopher Popfinger, Cristian Pérez de Laborda, and Stefan Conrad

Institute of Computer Science
Heinrich-Heine-Universität Düsseldorf
D-40225 Düsseldorf, Germany
{popfinger, perezdel, conrad}@cs.uni-duesseldorf.de

**Abstract.** Collaborative work requires, more than ever, access to data located on multiple autonomous and heterogeneous data sources. The development of these novel information platforms, referred to as information or data grids, and the evolving databases based on P2P concepts, need appropriate modeling and description mechanisms. In this paper we propose the Link Pattern Catalog as a modeling guideline for recurring problems appearing during the design or description of information grids and P2P networks. For this purpose we introduce the Data Link Modeling Language, a language for describing and modeling virtually any kind of data flows in information sharing environments.

## 1 Introduction

With the rise of filesharing systems like Napster or Gnutella the database community started to seriously adopt the idea of P2P systems to the formerly known loosely coupled databases. While the original systems were only designed to share simple files among a huge amount of peers, we are not restricted to these data sources any more. New developments allow peers to share virtually any data, no matter if it is originated from a relational, object-oriented, or XML database. In fact, the data may still come from ordinary flat files.

Apparently we have to deal with a very heterogeneous environment of data sources sharing data, referred to as an information or data grid [4]. If we allow participants to join or leave information grids at any time (e.g. using P2P concepts [3]), we must take a constantly changing constellation of peers into account. Any information grid built up by these peers can either evolve dynamically or be planned beforehand. In both cases we need a concept in order to describe and understand the interactions among the peers involved. Having such a mechanism, we could not only detect single data exchanges, but even model and optimize complex data flows of the entire system.

In this paper we adopt commonly used methods for designing data exchanges among peers as *Link Patterns*, suitable especially for information grids and P2P networks. Analogous to the intention of the Design Pattern Catalog used for object-oriented software development [8] we want to provide modeling guidelines for engineers and database designers, engaged in understanding, remodeling, or building up an information grid. Thus information grid architects are provided with a common vocabulary for design and communication purposes.

Up to now data flows in information grids were designed without having a formal background leading to individual solutions for a specific problem. These were only known to a circlet of developers involved into that project. Other designers, engaged with a similar problem would never get in contact with these results and thus make the same mistakes again. Different modeling techniques make it difficult to exchange successfully implemented solutions.

Link Patterns do not claim to introduce novel techniques for sharing, accessing, or processing data in shared environments, but a framework for being able to understand, describe, and model their data flows. They provide a description of basic interactions between data sources and operations on the data exchanged, resulting in a catalog of reusable conceptual units.

A developer may choose Link Patterns to model and describe complex data flows, to identify a single point of failure, or to avoid or consciously insert redundant data exchanges. The composition of Link Patterns is an essential feature of our design method. It gives us the possibility to represent a structured visualization not only of single data linkages, but of the entire information platform.

The remainder of this paper is organized as follows. In section 2 we introduce DLML, a language for modeling data flows, followed by a structural description of the Link Patterns in section 3. Section 4 specifies the Link Pattern Catalog, followed by an example. Section 6 catches up some related work and section 7 concludes.

## 2 The Data Link Modeling Language (DLML)

### 2.1 Introduction

The Data Link Modeling Language (DLML) is based on the *Unified Modeling Language* (UML) [8] notation, but slightly modifies existing components, adds additional elements, and thus extends its functionality. It is a language for modeling, visualizing, and optimizing virtually any kind of data flows in information sharing environments.

**Modeling:** DLML is a language, suitable for modeling, planning, and re-engineering data flows in information sharing environments, e.g. information grids, systematically. A Data Link Model built up using this language reflects the logical and not the physical structure of the entire system. It enables the developer to specify the properties and the behavior of existing and novel systems, in order to describe and understand their basic functionalities.

**Visualizing:** Visualizing data flows is an important assistance in understanding the structure and behavior of an information platform. The impact of ER [13] and UML has proven, that a system is easier to grasp and less error-prone, if a graphical visualization technique is provided, which uses a well-defined set of graphical symbols, understood by a broad community. Especially within the analysis of systems with distributed information, it is favorable to have a method, suitable for drawing up a map of relationships between the participating peers, in order to depict global data flows.

**Optimizing:** Besides the modeling and visualization of an information sharing environment, DLML can be useful to optimize the whole distributed data management. Redundant data flows and data stocks can systematically be detected and removed, leading to a higher performance of the entire system. Of course, redundancy may explicitly be wanted, in order to achieve a higher fail-safety or a faster access to the data.

Due to the characteristics mentioned above, the Data Link Modeling Language is especially suitable for visualizing data flows in distributed information grids. It may furthermore be employed to model data management in enterprise information systems, data integration and migration scenarios, or data warehouses, i.e. wherever data has to be accessed across multiple different data sources.

## 2.2 Components

Since DLML is based on UML, its diagrams are constructed in an analogous manner, using a well-defined set of building blocks according to specific rules. The following components may be used in DLML (Fig. 1) to build up a Data Link Model:

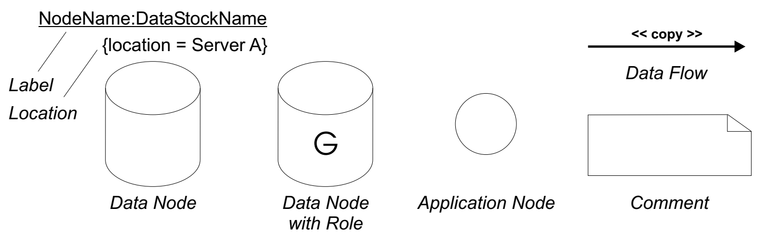

**Fig. 1.** DLML Components

**Nodes:** Nodes are data sources, data targets, or applications, usually involved in a data exchange process. They may either be isolated or connected through at least one *data flow*. A data source may be a database (e.g. relational), a flat file (e.g. XML), or something similar, offering data, whereas a data target receives data and stores it locally. An application is a software unit, which accesses or generates data, without maintaining an own physical data stock. Physical data stocks are represented in DLML by *Data Nodes*, applications by *Application Nodes*.

**Label:** Each node can have a label. It consists of generally two parts separated by a colon: the *node name* and the *data stock name* or *application name* respectively. The data stock name identifies the combination of data and schema information stored at this node. If this data is replicated as an exact and complete copy to another node, the data target has to use the same data stock name. The application is identified by the application name. Analogous to the data stock name, any further instances of the same application have the same application name. In both cases we use the node name to distinguish nodes with the same data stock or application name. Otherwise the node name is optional.

**Location:** The optional location tagged value specifies the physical location of the node. It either specifies an IP address, a server name, or a room number, helping the developer to locate the Data or Application Node.

**Role:** A node providing a certain functionality on the data processed, may have a functional role (e.g. filtering or integrating data). This role will usually be implemented as a kind of application, operating directly on the incoming or outgoing data. The name of the role or its abbreviation is placed directly inside the symbol of the node. This information is not only useful for increasing the readability of the model, but also for being able to identify complex relationships.

**Data Flow:** The data exchange between exactly one data source and one data target is called data flow. The arrow symbolizes the direction, in which data is being sent. A node may have multiple incoming and outgoing data flows. Optionally each data flow may be labeled concerning its behavior, i.e. if the data is being replicated (`<<copy>>`) to the data target or if it is just accessed (`<<access>>`). If data is being synchronized, both data flow arrows may be replaced by one single arrow with two arrowheads.

**Comment:** A comment may be attached to a component, in order to provide additional information about a node or a data flow. These explanations may concern a node's role, filter criteria, implementation hints, data flow properties, or further annotations important for the comprehension of the model.

## 2.3 Example

We now illustrate the usage of the Data Link Modeling Language with a simplified example. Consider a worldwide operating wholesaler, with an autonomous overseas branch. The headquarters is responsible for maintaining the product catalog (`hq:products`) with its price list, while the customers database (`:customers`) is administrated by the branch itself (Fig. 2).

The overseas branch is connected to the headquarters by a dial-up connection, not sufficient for accessing the database permanently. For this reason, the product catalog is replicated to the branch twice a day (`branch:products`), where the data may be accessed by the local employees. The branch management uses a special application (`:managementApp`) to access both data stocks in order to generate the annual report for the headquarters.

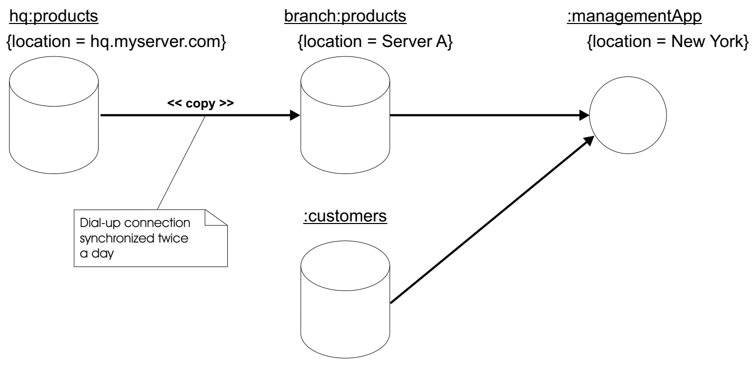

**Fig. 2.** DLML Example

## 3 Link Patterns

In order to be able to provide a catalog of essential Link Patterns it is necessary to understand what a Link Pattern is. Therefore we present the elements a Link Pattern is composed of, including its name, its classification, or its description. For graphical representation we use the Data Link Modeling Language, specified above.

### 3.1 Elements of a Link Pattern

In this section we present the description of the Link Pattern structure. It is based on the Design Pattern Catalog of Gamma et al. [8], which has reached great acceptance within the software engineering community. Thus a developer is able to quickly understand and adopt the main concept of each Link Pattern for his own purposes. Each Link Pattern is described by the following elements:

**Name:** The name of a Link Pattern is its unique identifier. It has to give a first hint on how the pattern should be used. The name is substantial for the communication between or within groups of developers.

**Classification:** A Link Pattern is classified according to the categories described in section 3.2. The classification organizes existing and future patterns depending on their functionality.

**Motivation:** Motivating the usage of the pattern is very important, since it explains the developer figuratively the basic functionality. This is done using a small scenario, which illustrates a possible application field of the pattern. Therewith the developer is able to understand and follow the more detailed descriptions in the further sections.

**Graphical Representation:** The most important part of the pattern description is the graphical representation. It is a DLML diagram and describes the composition and intention of the pattern in an intuitive way. The developer is advised to adopt this representation, wherever he has identified the related functionality in his own information grid model.

**Description:** The composition of the Link Pattern is described in-depth in this section, including every single component and its detailed functionality. The explanation of the local operations on each node and data flows between the components involved, points up the intended functionality of the whole pattern described. This description shall give the user both, a guidance through the identification process and instructions for its proper usage.

**Challenges:** Besides the general instructions given in the prior section, this section shall give hints for sources of error in the implementation process of this pattern. The developer shall get ideas, of how to identify and avoid pitfalls, arising in a certain context (e.g. interaction with other Link Patterns).

## 3.2 Classification

A classification of the Link Pattern Catalog shall provide an organized access to all Link Patterns presented. Patterns situated in the same class have similar structural or functional properties, depending on the complexity of their implementation. Although a categorization of a very limited number of patterns may seem superfluous, we have decided to include this into our Link Pattern Catalog, since it shall help developers to allocate and evaluate the pattern required. Furthermore it should stimulate the developer to find and rate novel patterns, not yet included in the catalog.

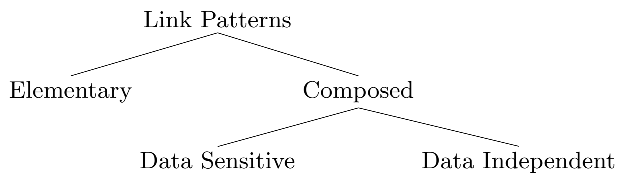

**Fig. 3.** Link Pattern Catalog Classification

Figure 3 depicts the classification of our Link Pattern Catalog we have chosen. The patterns presented can be divided into two main categories, *Elementary Link Patterns* and *Composed Link Patterns*. In fact this classification is not completed, but shall provide a starting point for further extension.

**Elementary Link Pattern:** An Elementary Link Pattern is the smallest unit for building up an information grid model. It consists of exclusively one single node and at least one data flow connected to it. Each Data Link Model is composed of several Elementary Link Patterns, linked together with data flows in an appropriate way. Please note, that a single Elementary Link Pattern is not yet a reasonable Data Link Model, since any data flow must have at least one node offering data and one node receiving data.

Elementary Link Patterns are easy to understand and easy to implement, since they concern only a single node, a small set of data flows, and do not include basically any data processing logic. It must be pointed out, that the

Elementary Link Patterns consist only of two main patterns, the *Basic Data Node* and the *Basic Application Node*, and its derivatives (e.g. Publisher and Generator, discussed in section 4).

**Composed Link Pattern:** Composed Link Patterns are built up by combining at least two Elementary Link patterns in a specific way, in order to realize a particular functionality. A Composed Link Pattern may hereby be composed out of both, Elementary or other Composed Link Patterns. A pattern has to represent a prototype or solution for a recurring sort of problem. Please keep in mind, that an arbitrary combination of different patterns will not automatically lead to a reasonable Composed Link Pattern.
In contrast to the Elementary Link Patterns, we have to deal in this context with a more complex kind of patterns. They do not only include more nodes, but may even represent a quite sophisticated way of linking them. Besides, each node may additionally process the data received or sent. The fact, that it may act differently depending on the data involved, is an essential property of Composed Link Patterns and justifies the creation of two subclasses:

**Data Sensitive Link Pattern:** As soon as a node included in a Composed Link Pattern acts depending on the data it processes, the entire pattern is called a Data Sensitive Link Pattern. This data processing logic implemented on such a node may depend on and be applied to incoming and/or outgoing data. The operations of this application can either create, alter, or filter data.

**Data Independent Link Pattern:** Any Composed Link Pattern, not classified as Data Sensitive, belongs to this class. In contrast to the patterns described above, data is not being modified, but sent or received as is. A rather crucial topic is the topology of the nodes and data flows involved, which is most relevant for the creation and functionality of this kind of patterns.

### 3.3 Usage

This section describes how Link Patterns can be useful to develop, maintain, analyze, or optimize both, straightforward and complex data flows in information grids. There are basically two methods, how Link Patterns can improve the work of developers:

**Analyzing existing systems:** Many existing information grids have arisen during the years without being planned centrally or consistently. Even if they were planned initially, they usually tend to spread in an uncontrolled way. In such an environment it is vital to have supporting tools, helping to understand and later optimize an existing system.
First of all a map or model of the existing system has to be created, e.g. with DLML presented in section 2. Afterwards we examine successively smaller parts of the model, in order to match them to existing Link Patterns of the Catalog. As a result we get a revised model containing basic information on the composition and functionality of subsystems, including their data

processing and data flows. With this information in mind, we are now able to derive information on data flows and interaction of nodes inside the Data Link Model. This enables us to perform optimizations like detecting and eliminating vulnerabilities or handling redundancies.
Link Patterns may thus not replace human expertise for understanding existing information grids, but give support in the process of recognizing global data flows and therewith interpret the purpose of the entire system.

**Composing new models:** As already mentioned a Link Pattern may not only improve the process of understanding an existing information grid, but is also a support for modeling new systems. An information grid architect needs to have a clear idea of what the system should do. Depending on the data sources available, the local requirements on the nodes, and the results he wants to achieve, he can combine nodes and data flows, according to Link Patterns, until the entire system realizes the intended functionality. Link Patterns hereby guarantee a common language, understood by other developers, not yet involved in the modeling. Each developer is thus able to quickly get a general idea of the system modeled at any time. Furthermore they accelerate the development process, since they provide well tried solutions for recurring problems, leading to a performant system of high quality.

## 4 Link Pattern Catalog

In this section we finally give an introduction into the Link Pattern Catalog. This includes a graphical overview over the main Link Patterns in DLML, as well as a detailed description of selected patterns. As mentioned beforehand the Link Patterns can be classified according to the classification presented in section 3.2. Since any Composed Link Pattern either belongs to the Data Sensitive or to the Data Independent Link Patterns, we organize the catalog as follows:

**Elementary Link Patterns**
The Elementary Link Patterns are the basic building blocks of a Data Link Model. They consist of the two basic patterns, described below, and its derivatives. All Elementary Link Patterns are depicted in Figure 4.

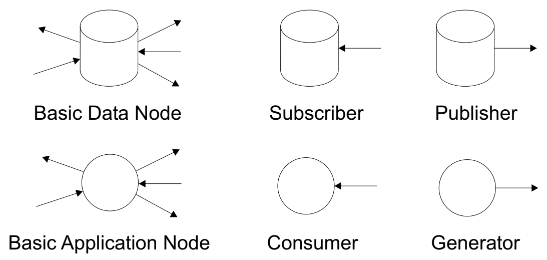

**Fig. 4.** Elementary Link Patterns

**Basic Data Node**

**Classification:** Elementary Link Pattern

**Motivation:** This pattern is one of the basic building blocks of a Data Link Model. Each incoming or outgoing data flow of a Data Node is modeled using this Link Pattern.

**Graphical Representation:** See Figure 4

**Description:** A Basic Data Node is a DLML Data Node, which receives data through incoming data flows, stores it locally, and simultaneously propagates data, held in its own data stock. If a Basic Data Node does only have outgoing or incoming data flows, it applies the *Publisher Pattern* or the *Subscriber Pattern* respectively. If it does neither have any incoming, nor any outgoing data flows, the Data Node is called *isolated*.

**Challenges:** One of the main challenges to take in this pattern is the proper coordination of incoming and outgoing data flows. At first all incoming data has to be stored permanently on the local data stock, without violating any constraints, before it may be propagated again to other nodes.

**Basic Application Node**

**Classification:** Elementary Link Pattern

**Motivation:** This pattern is one of the basic building blocks of a Data Link Model. All applications, relevant for a Data Link Model, are based on this pattern.

**Graphical Representation:** See Figure 4

**Description:** An application interacting with arbitrary Data or Application Nodes, is represented by this pattern. The application does not only receive, but also propagate data. If a Basic Application Node does only have outgoing or incoming data flows, it applies the *Generator Pattern* or the *Consumer Pattern* respectively. If it does neither have any incoming nor any outgoing data flows, the Application Node is called *isolated*.

**Challenges:** Propagated data can either be received or generated. All data manipulations on incoming data, which have to be propagated, have to be processed in real-time, without storing data locally.

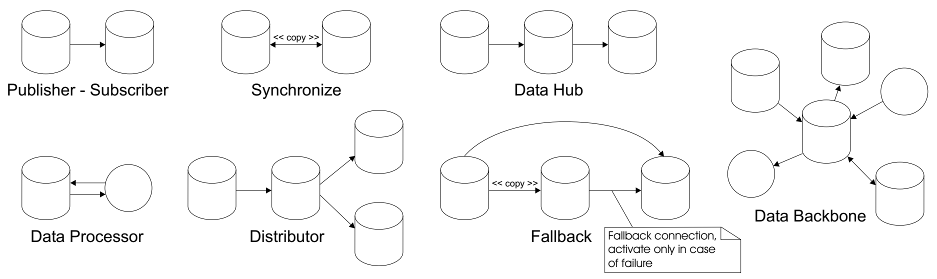

**Fig. 5.** Data Independent Link Patterns

**Data Independent Link Patterns**
The Data Independent Link Patterns belong to the Composed Link Patterns. These patterns describe a functionality, which only depends on their structure, i.e. the way nodes and data flows are combined. A graphical overview of the patterns in this class is given in Figure 5, of which the *Data Backbone* is described exemplarily.

**Data Backbone**
**Classification:** Data Independent Link Pattern
**Motivation:** A Data Backbone is used, wherever a centralization of data sharing or data access has to be realized. This is typically required, if data stocks are re-centralized, a central authority wants to keep track on all data flows, or data exchanges have to be established among multiple data stocks and applications.
**Graphical Representation:** See Figure 5
**Description:** The Data Backbone Pattern consists of several nodes, linked together in a specific way. A designated node, called Data Backbone, is either data source or data target for all data flows in this pattern. All nodes, including the Data Backbone itself, can be data stocks or applications. Data is always propagated from data sources to the Data Backbone, where it may be accessed or propagated once again to other target nodes. Direct data flows between nodes, which are not the Data Backbone, are avoided.
**Challenges:** Since the Data Backbone is involved in all data flows, it has a crucial position in this part of the information grid. Thus, a Data Backbone node has to provide a high quality of service, concerning disk space, network connection, and processing performance. If the quality of service required cannot be provided, the Data Backbone may easily become a bottleneck. Furthermore a breakdown of this node could lead to a collapse of the entire data sharing infrastructure, which makes it to a single point of failure.

**Data Sensitive Link Patterns**
Contrary to the Data Independent Link Patterns, the patterns described in this section are not only classified according to their structural properties, but particularly because of their data processing functionality. A graphical representation of these Data Sensitive Link Patterns can be found in Figure 6, while a detailed description is only given for the *Gatekeeper Pattern*.

**Gatekeeper**
**Classification:** Data Sensitive Link Pattern
**Motivation:** A Gatekeeper is used to control data flows according to specific rules (e.g. Access Control Lists), stored separately from the data processed. It is responsible for providing the target nodes with the accessible data required. The application of this pattern is not limited to data security matters. It may actually be applied to any node, which has to supply different target nodes with specific (e.g. manipulated or filtered) data flows.

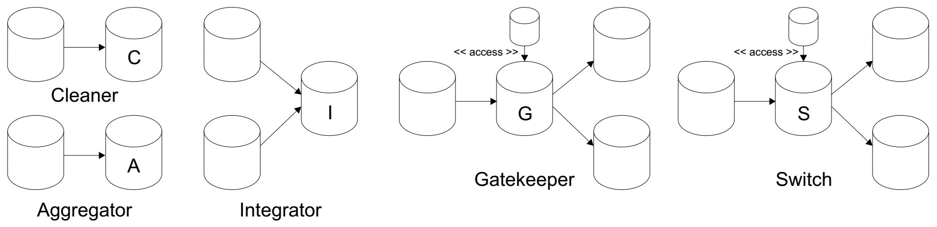

**Fig. 6.** Data Sensitive Link Patterns

**Graphical Representation:** See Figure 6

**Description:** A Gatekeeper is a designated node, which distributes data according to specific rules, eventually stored separately. Local or incoming data of a Gatekeeper is accessed by target nodes. Before this access can be admitted, the Gatekeeper has to check the permissions. Thus, corresponding to the rules processed, neither all data stored in the Gatekeeper, nor all data requested by the target nodes has to be transmitted.

**Challenges:** The rules and techniques, which are used by the Gatekeeper in order to secure access to the data, have to be robust and safe. The Gatekeeper needs a mechanism to identify and authenticate the source and target nodes (e.g. IP address, public key, username and password, or identifiers [11]), which may be stored in a separated data stock. Due to its vital position in the exchange process, this information has to be protected from unauthorized access. The Gatekeeper must be able to rely on the correctness, authenticity, and availability of the rules required.

## 5 Example

This section provides an example of how to model a new information grid of a worldwide operating company. The headquarters of the company are located in New York. It has additionally branches in Düsseldorf (head office of the European branches), Paris, Bangalore, and Hong Kong. Each branch maintains its own database containing sales figures, collected by local applications. For backup and subsequent data analysis, this data has to be replicated to the headquarters. Additionally, the Düsseldorf branch needs to be informed about the ongoing sales activities of the Paris branch. To simplify the centralized backup, the company has decided to forbid any data exchanges between the single branches.

The central component of this infrastructure is the backup system in New York. It collects the sales data from all branches, without integrating them. Additionally it provides the Düsseldorf branch with all the information required from Paris. Since the headquarters in New York want to analyze the entire data stock of the company, a data warehouse, based on the data of the backup system, is set up. Having a certain local autonomy, the data provided by the European branches and the remaining branches have some structural differences. For this reason, the data has to be integrated prior to the aggregation required for the data warehousing analysis.

Using the Link Patterns proposed in this paper, we are now able to model the enterprise information grid as depicted in Figure 7.

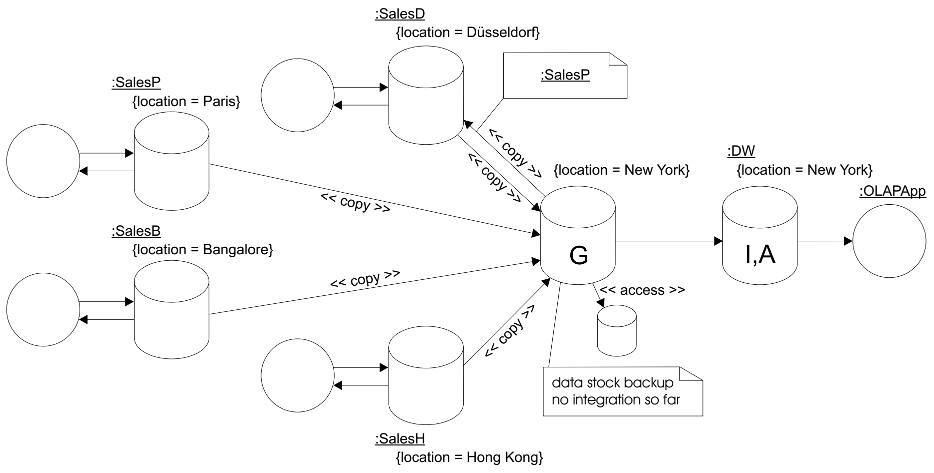

**Fig. 7.** Example using Link Patterns

The local applications, which maintain the local sales databases, are modeled using the *Data Processor*. This data is replicated to the backup system in New York, realized as a *Gatekeeper*. It thus controls the data flows from the branches to the data warehouse and to the Düsseldorf branch. It must be guaranteed, that the data targets get only their designated data, i.e. neither data from Bangalore, nor from Hong Kong is accessible for the European head office in Düsseldorf. The data warehouse is realized by a node, which integrates several data sources using common integration strategies (*Integrator Pattern*) and aggregates the data afterwards (*Aggregator Pattern*), in order to provide OLAP applications with a homogenous data stock.

Please keep in mind, that the Data Link Model presented in Figure 7 reflects the logical structure of the information platform, not the physical. This means, that the nodes of the model do not have to be located on different machines.

## 6 Related Work

Data Flow analysis and modeling has been a focus of researchers for decades. Earlier work concentrates mainly on data flows in computer architectures and software components (e.g. [15, 5]). Later on, data flows were also used for query processing and optimization in database systems. For instance, Teeuw and Blanken [14] compare control versus data flow mechanisms controlling the execution of database queries on parallel database systems.

Dennis and Misunas present in [6] a Basic Data-Flow language, which expresses graphically the data dependencies within a program. In this data flow graph model, instructions are represented by nodes and paths stand for data or control flows. Although this language was originally designed for software development, it may be seen as an early forerunner, in designing data flows among different data sources. A specialized data flow graph is introduced by Eich and Wells [7], which can be used for scheduling database queries within multiprocessor environments or databases distributed over a network [1]. Thus, both approaches apply data flow concepts to database processing.

The Link Patterns are tightly coupled to the Design Patterns of the object-oriented software design [8, 2] and Enterprise Application Integration (EAI) [10], since they represent prototypes or solutions for recurring problems. Contrary to these patterns, Link Patterns are not intended to solve recurring problems in software design or EAI, but to provide modeling and description guidelines for information grids, focusing exclusively on data flows.

As a possible application field of our Link Patterns we suggest modeling or visualizing information grids, i.e. heterogeneous environment of data sources sharing data, or modern information infrastructures, based on P2P concepts (e.g. [9] or [12]).

## 7 Conclusion and Future Work

In this paper we have presented Link Patterns as guidelines for modeling and describing data flows between nodes in information sharing environments. The Link Pattern Catalog consists of prototypes or solutions for recurring problems and therewith supports developers to model, describe, and understand complex information grids. Furthermore the Link Patterns provide a common vocabulary for design and communication purposes, enabling developers to exchange successfully implemented solutions.

Additionally we have introduced the Data Link Modeling Language (DLML) for modeling, visualizing, and optimizing data flows, especially suitable for information grids. This language based on UML consists of a well-defined set of building blocks, representing data nodes, application nodes and data flows between them. They can be combined according to specific rules, to build up the Data Link Model of an information sharing environment.

The concepts we have presented in this paper are ideal to generate a static model of data and application nodes with their corresponding data flows. In future work we have to consider dynamically changing and evolving environments, in which nodes constantly join or leave the grid. This may not only affect the Link Pattern Catalog, but also the Data Link Modeling Language. Furthermore the Catalog has to be enhanced, in order to include novel Link Patterns, not yet identified. The entire Link Pattern Catalog shall provide developers with an extensive reference guideline for modeling information sharing environments.

## References

1. Lubomir Bic and Robert L. Hartmann. AGM: A Dataflow Database Machine. *ACM Transactions Database Systems*, 14(1):114–146, 1989.

2. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., 1996.
3. Mario Cannataro and Domenico Talia. Semantics and knowledge grids: Building the next-generation grid. *IEEE Intelligent Systems*, 19(1):56–63, 2004.
4. Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven 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.
5. Lori A. Clarke, Andy Podgurski, Debra J. Richardson, and Steven J. Zeil. A Formal Evaluation of Data Flow Path Selection Criteria. *IEEE Trans. Softw. Eng.*, 15(11):1318–1332, 1989.
6. Jack B. Dennis and David P. Misunas. A Preliminary Architecture for a Basic Data-Flow Processor. In *Proceedings of the 2nd Annual Symposium on Computer Architecture*, pages 126–132. ACM Press, 1975.
7. Margaret H. Eich and David L. Wells. Database Concurrency Control Using Data Flow Graphs. *ACM Transactions Database Systems*, 13(2):197–227, 1988.
8. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.
9. Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proceedings of the twelfth international conference on World Wide Web*, pages 556–567, Budapest, Hungary, 2003.
10. Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2003.
11. Cristian Pérez de Laborda and Stefan Conrad. A Semantic Web based Identification Mechanism for Databases. In *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003), Hamburg, Germany, September 15-16, 2003*, volume 79 of *CEUR Workshop Proceedings*, pages 123–130. Technical University of Aachen (RWTH), 2003.
12. Cristian Pérez de Laborda, Christopher Popfinger, and Stefan Conrad. DÍGAME: A Vision of an Active Multidatabase with Push-based Schema and Data Propagation. In *Proceedings of the GI-/GMDS-Workshop on Enterprise Application Integration (EAI'04)*, volume 93 of *CEUR Workshop Proceedings*, 2004.
13. Peter Pin-Shan Chen. The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.
14. Wouter B. Teeuw and Henk M. Blanken. Control versus Data Flow in Parallel Database Machines. *IEEE Transactions on Parallel and Distributed Systems*, (4):1265–1279, 1993.
15. Elizabeth Winey. Data Flow Architecture. In *Proceedings of the 16th annual Southeast regional conference*, pages 103–108. ACM Press, 1978.