Building Scalable Electronic Market Places using HyperQuery-Based Distributed Query Processing *

Alfons Kemper and Christian Wiesner
Universität Passau, Lehrstuhl für Informatik, 94030 Passau, Germany, e-mail: <last name>@db.fmi.uni-passau.de

Abstract. Flexible distributed query processing capabilities are an important prerequisite for building scalable Internet applications, such as electronic Business-to-Business (B2B) market places. Architecting an electronic market place in a conventional data warehouse-like approach by integrating *all* the data from *all* participating enterprises in one centralized repository incurs severe problems: stale data, data security threats, administration overhead, inflexibility during query processing, etc. In this paper we present a new framework for dynamic distributed query processing based on so-called *HyperQueries* which are essentially query evaluation sub-plans "sitting behind" hyperlinks. Our approach facilitates the pre-materialization of static data at the market place whereas the dynamic data remains at the data sources. In contrast to traditional data integration systems, our approach executes essential (dynamic) parts of the data-integrating views at the data sources. The other, more static parts of the data are integrated à priori at the central portal, e.g., the market place. The portal serves as an intermediary between clients and data providers which execute their sub-queries referenced via hyperlinks. The hyperlinks are embedded as attribute values within data objects of the intermediary's database. Retrieving such a virtual object will execute the referenced HyperQuery in order to materialize the missing data. We illustrate the flexibility of this distributed query processing architecture in the context of B2B electronic market places with an example derived from the car manufacturing industry.

Based on these HyperQueries, we propose a reference architecture for building scalable and dynamic electronic market places. All administrative tasks in such a distributed B2B market place are modeled as Web services and are initiated decentrally by the participants. Thus, sensitive data remains under the full control of the data providers. We describe optimization and implementation issues to obtain an efficient and highly flexible data integration platform for electronic market places. All proposed techniques have been fully implemented in our QueryFlow prototype system which served as platform for our performance evaluation.

Keywords: Distributed Databases, Portal, Product Catalog, E-Business, Dynamic Query Processing, Reference Architecture, Data Integration

* This work was supported by the German National Research Council (DFG) under Contract Ke 401/7-1 and Ke 401/7-2 and by SAP. Some excerpts of this work appeared in the VLDB 2001 conference publication [34]. A demonstration of the implemented QueryFlow system was presented at the EDBT 2002 conference [66].

Table of Contents

1. Introduction

Electronic market places and virtual enterprises have become very important applications for query processing [6, 29]. Building a scalable electronic Business-to-Business (B2B) market place with hundreds or thousands of participating suppliers requires highly flexible, distributed query processing capabilities. Bakos [5] characterizes electronic market places as "inter-organizational information systems that allow the participating buyers and sellers to exchange information about prices and product offerings". Thus, the market place in fact constitutes a data integration portal, as customers pose queries and suppliers provide information. Architecting such an electronic market place as a data warehouse by integrating *all* the data from *all* participants in *one* centralized data repository incurs severe problems:

- *Security and privacy violations:* The participants of the market place have to relinquish the control over their data and entrust sensitive information, e.g., pricing conditions, to the market place.
- *Coherence problems:* The coherence of highly dynamic data, such as availability and shipping information, may be violated due to outdated materialized data in the market place's data warehouse.
- *Schema integration problems:* Using the warehouse approach all relevant data from all participants have to be converted à priori into the same format. Often, it would be easier to leave the data inside the participants' information systems, e.g., legacy systems, and apply particular local wrapper/transformer operations. This way, data is only converted *on demand* and the most recent coherent state of data is returned.
- *Fixed query operators:* In a fully integrated electronic market place all information is materialized at the central data warehouse. This is often not desirable in such complex applications like electronic procurement/bidding. For example, in pricing offers one would like to have different and flexible approaches, e.g.:
  - fixed pricing via materialized data
  - operators which calculate the prices based on many local and global parameters (identity of the consumer, availability, local plant utilization, sub-contractor prices, etc.)
  - even human interaction during the processing of such complex e-procurement queries is desirable. In some participating enterprises the pricing could be done by humans via an interactive "query operator".

We propose so-called *HyperQueries* to architect highly flexible distributed query processing systems. HyperQueries are essentially query evaluation sub-plans "sitting behind" hyperlinks . This way, a portal like an electronic market place can be built as an intermediary between the client (issuing a query) and the providers executing their subqueries referenced via hyperlinks. The hyperlinks to the HyperQueries are embedded as attribute values within data objects (i.e., tuples) of the intermediary's database. Retrieving such a (partially) virtual object automatically initiates the execution of the referenced HyperQuery in order to materialize the entire object. Thus, sensitive data can remain under the full control of the data providers. Instead of replicating the data at the intermediary, only the hyperlink is embedded.

1.1. Contributions

The HyperQuery framework allows to blur the distinction between the allocation schema and the data—as it is found in clear separation in traditional distributed databases. Thereby, the distribution of query execution plans is highly dynamic and based on attribute values obtained during query processing.

Based on HyperQueries we present a reference architecture for building scalable, distributed B2B market places. We loosely couple participants and obtain a "market place federation" where new participants can easily join the market place. The integration of existing systems, support of open standards, security, fault tolerance, and administration of the market place are further important aspects of our approach. The participants are able to manage their data themselves without the interference of the administrator of the market place. In our prototype system, these administrative tasks are supported by web services. Our reference architecture is fully implemented and is based on open standards.

In this paper we present a query evaluation technique for distributed data integration systems with the example application of B2B market places. Schema integration is beyond the scope of this paper. We assume, that this has already happened, i.e., schemas from multiple data sources using different data models, schemas, vocabularies, and ontologies have been integrated leading to one central *mediated schema* and *mapping rules* which define how to map information of the data sources into the mediated schema. The primary goal of this paper is to present both a novel architecture and query processing techniques for the execution of queries against a mediated schema. Using our approach mediated data can be divided into a static part which is replicated centrally, e.g., at the portal or market place, and a dynamic part remaining at the data sources.

1.2. Related Work

Distributed database systems have been studied since the late seventies in projects like System R* [67], SDD-1 [9], and Distributed Ingres [58]. All these approaches extend single side DBMSs to manage relations that are spread over the sites in a computer network in a seamless manner. Middleware systems such as TSIMMIS [47], DISCO [62], and Garlic [22, 30] have been used to overcome the heterogeneity faced when data is dispersed across different data sources. Wiederhold [65] describes and classifies methods to access data in a three-layer, mediated architecture. Crescenzi et al. [17] show the automatic generation of wrappers for HTML pages. These techniques can be adapted within our approach but are (taken alone) too rigid to perform dynamic query execution. Yang and Papazoglou [70] present a two-phase optimizer to reduce the search space and pay attention to dynamic costs for accessing Web sources of limited query capability. Papakonstantinou [46] addresses the fusion of information from heterogeneous information sources by removing redundancies and resolving inconsistencies. Sheth and Larson [56] discuss reference architectures for federated DBMSs from system and schema viewpoints. All these middleware solutions require the administrators to manually install all the necessary functionality for query processing. Newer architectures such as MOCHA [49] and ObjectGlobe [11] integrate dispersed data sources and provide the autonomous loading of functionality from an external code repository. Finally, [36] gives a survey of distributed query processing techniques.

There has been a lot of work on schema and data integration and many systems have been proposed within this context. Lenzerini [37] and Levy [38] provide surveys on data integration systems and associated query evaluation techniques. Basically two mapping approaches between the sources and the mediated schema are identified. In the *Global-as-View*-approach (GAV) for each relation R in the mediated schema, a view in terms of the source relations is defined which specifies how to obtain R's tuples from the sources. TSIMMIS [47] is one example system, that uses the GAV approach. In the *Local-as-View*-approach (LAV) for each data source S, a view in terms of the mediated schema relations is written. This view describes the information that can be retrieved from S in terms of the mediated schema relations. Information Manifold [39, 40] and SIMS [3] are examples for LAV systems.

Our approach adapts the GAV approach, i.e., the market place defines a mediated schema and HyperQueries specify how data values of the mediated schema can be obtained from the sources. But in contrast to traditional data integration systems, data providers create their HyperQueries autonomously. HyperQueries reside at the data sources and are executed under the full local control of the data sources. In our

framework (the static) parts of the mediated schema can be materialized centrally, e.g., at the market place, while other (dynamic) parts remain at the suppliers and are determined on demand. This way, both static data (such as descriptions, comments, and images) and highly dynamic data (e.g., prices and availability) are always up-to-date and can be queried in the standard (central data warehouse-like) way.

InfoSleuth [31, 44], SIMS [3], Observer [41], and Ariadne [35] have put their focus on utilizing ontologies. The approach of Goldman and Widom [20] combines the query facilities of traditional databases with existing search engines on the Internet whereby the execution of queries is sped up by parallelizing the Web search. Cohera [25] is based on the economic model of Mariposa [60] and integrates heterogeneous data sources using replication tools. Tukwila [28] provides adaptive execution and scalability for a large number of data sources across intranets and the Internet. Solutions to warehouse XML data have been proposed, e.g., the Xyleme system [68]. The Nimble XML data integration system [18] queries both materialized data in a central warehouse and heterogeneous data sources on demand. When integrating data from multiple data sources, inconsistencies can occur. Rahm and Do [48] describe common problems in data cleaning. Novel approaches such as PIAZZA [23] and Edutella [42] use P2P technology for the discovery and integration of heterogeneous data sources.

Our HyperQueries bear some similarity with Stonebraker et al.'s work on "QUEL as a Datatype" [59], which, however, was restricted to stored sub-queries in centralized database systems. HyperQueries are also related to pointer-based join methods [55] of relational and object-relational database systems, as the hyperlinks can be viewed as pointers to data at remote sites. Keller et al. [33] describe the so-called *object assembly* which optimizes the order in which objects are read from disk or retrieved from remote servers in a distributed system in order to reduce I/O cost. Object assembly is specifically designed to assemble complex objects that are hierarchically composed of sub-objects.

Yang and Papazoglou [69] describe a reference architecture for interoperable e-commerce applications. Casati et al. [13] discuss requirements and challenges for e-business applications that support supply chain management and propose an architecture to meet these requirements. Virtual enterprises and B2B e-commerce environments present an important application domain for our new technique: the automobile industry's electronic market place endeavor "Covisint" [16] and SAP's "mySAP.com" [51] electronic market places are among the well known examples. Initiatives like Bea's WebLogic [8], Sun ONE [61], Microsoft .NET [43], or IBM WebSphere [64] show that Web service technology for application collaboration and integration gains increas-

ing attention in research and industry. Florescu and Kossmann [19] present a platform for web services based on an abstract programming language and supporting web conversations and Web service composition. Keidl et al. [32] introduce an architecture for dynamic selection and distribution of Web services.

ActiveXML [1, 2] is a recent related approach to harness Web services for data integration and is put to work in a Peer-to-Peer architecture: an XML document containing references to Web services is processed and the references are replaced by the results of calls to the corresponding Web services. This way, dynamic parts of the XML document are materialized on demand. Their technique is based on Web services and their focus has not been on query optimization as our HyperQueries, e.g., it is not possible to bundle multiple invocations of Web services and ActiveXML provides no pipelining.

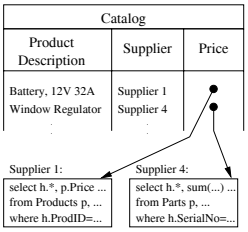
1.3. Running Example

As an example application we present a scenario derived from the car manufacturing industry. We assume a central market place and multiple suppliers and sub-contractors. The suppliers have registered their products at the central market place and sub-contractors on their part have registered their products at the suppliers. As this multi-layer portal structure may be arbitrarily deep, we obtain a hierarchical supply chain. A typical process of e-procurement to cover unscheduled demands is to query the market place for these products and to select the incoming offers based on price, terms of delivery, available quantity, rating, etc. The price of the needed products can vary by customer/supplier-specific sales discounts, the quantity of materials to be provided, plant utilization, etc. Therefore, the price cannot be a materialized attribute as in traditional query processing systems. Instead price is an individually calculated, dynamically changing attribute and a hyperlink to the supplier is contained where the actual price will be computed on demand.

In traditional distributed query processing systems such a query can only be executed if all local databases are replicated at the market place or a global schema and mapping rules onto the distributed data sources exists. In the first case, i.e., all data is materialized at the market place, coherence problems with stale data may be encountered. In the latter case, all the data remains at the data sources. Then, a query can only be processed at very high costs as the global query has to be re-written in terms of the view definitions and all the data (static and dynamic) has to be shipped to the market place. Considering a market place with hundreds of participating suppliers, one global query integrating the sub-queries for all participants would be too complex and error-prone,

```
select    c.ProductDescription, c.Supplier, c.Price
from      NeededProducts p, Catalog@MarketPlace c
where     p.ProductDescription = c.ProductDescription
order by c.ProductDescription, c.Price
```

(a) Example Query of the Car Manufacturer



(b) Hyperlinks Referencing HyperQueries

Figure 1. Running Example

i.e., if one supplier's host is unavailable, the whole query execution would fail.

Following our approach the suppliers register their products at the market place and specify the sub-plans to compute the price information at *their* sites. The calculation of the price can be arbitrarily complex and involve their sub-contractors, too. Figure 1(a) shows an SQL-like client's query, that returns the description of products, prices, and the suppliers of all needed products. The static attributes *ProductDescription* and *Supplier* are taken from the materialized data at the market place, the value of the virtual attribute *Price* is determined by evaluating the hyperlinks referencing HyperQueries at the suppliers' hosts (see Figure 1(b); the complete HyperQueries are shown in Figure 3).

1.4. Organization of the Paper

The rest of this paper is organized as follows. In Section 2 we define HyperQueries and Section 3 illustrates the execution of HyperQueries. In Section 4 the reference architecture of a scalable dynamic market place federation consisting of a market place, hierarchically organized suppliers, and customers is described. In Section 5 we show optimization techniques of the HyperQuery execution and discuss some details of our implementation—the *QueryFlow*[1] system. We demonstrate the scalability of our approach in Section 6 and give a summary and an outlook on future work in Section 7.

[1] The name of our system was derived from *query* processing and work*flow* systems because processing queries with HyperQueries bears some similarity with processing distributed workflows by routing documents to the appropriate tasks.

2. Syntax and Semantics of HyperQueries

In our framework for data integration we differentiate between static and dynamic data. The former is à priori materialized in the portal, e.g., the market place, whereas the latter information remains at the data providers. The dynamic parts are integrated by so called HyperQueries and hyperlinks referencing HyperQueries are embedded as virtual attributes into the mediated schema. The HyperQueries reside at remote hosts nearby the data sources. During evaluation of a query involving virtual attributes the sub-queries are executed to integrate data on demand and materialize the current values. A similar approach has been taken in "QUEL as a Data Type" [59], but we do not store the sub-plans, which are executed at data providers, within virtual attributes. In this section we specify HyperQueries and show how hyperlinks and HyperQueries can be incorporated into the database design of a portal.

2.1. Metadata for HyperQuery Processing

Virtual attributes encapsulate hyperlinks and the results of the corresponding sub-plans as attribute values in a database table or as elements in an XML document. During evaluation the hyperlinks are replaced by the result values of the HyperQueries. The schema of these result values is specified by the market place in the mediated schema and is publicly available. Furthermore, the market place defines so-called *application-specific parameters*. These application-specific parameters are additional input parameters for the HyperQueries, are given by the client, and can be used within the HyperQueries to calculate the actual value of the virtual attributes. One object of the mediated schema can contain multiple virtual attributes that reference different sub-plans on possibliy different hosts.

2.2. Specification of Hyperlinks

To obtain general locators that are independent of the physical location of the referenced sub-plans, we define the following Uniform Resource Identifier (URI) schema for the specification of hyperlinks that reference HyperQueries:

```
<hqschema> ::= "hq://"<HostDNS>"/"<PathToPlanId>"?"<ParamList>
<ParamList>::= <ParamName>"="<ParamValue>
               {"&"<ParamName>"="<ParamValue>}
```

Here, the components have the following meaning: `hq` denotes our HyperQuery protocol, `<HostDNS>` is the DNS name of the host, on which the sub-plan is stored and executed, and `<PathToPlanId>` is the name of the stored sub-plan at the data provider. `<HostDNS>` and `<PathToPlanId>` are referred to as URI prefix. The subsequent

```
<Catalog>
   <Article ID="1">
      <ProductDescription>Battery, 12V 32A</ProductDescription>
      <Supplier>Supplier 1</Supplier>
      <Price isVirtual="true">
         hq://supplier1.com/Electrical/Price?ProdID=CB1232
      </Price>
   </Article>
   <Article ID="2">
      <ProductDescription>Battery, 12V 55A</ProductDescription>
      <Supplier>Supplier 1</Supplier>
      <Price isVirtual="true">
         hq://supplier1.com/Electrical/Price?ProdID=CB1255
      </Price>
   </Article>
   <Article ID="3">
      <ProductDescription>Tires 175/65TR14</ProductDescription>
      <Supplier>Supplier 2</Supplier>
      <Price isVirtual="true">
         hq://supplier2.com/Price?ProdKey=175/65TR14
      </Price>
   </Article>
   <Article ID="4">
      <ProductDescription>Spark Plug VX</ProductDescription>
      <Supplier>Supplier 3</Supplier>
      <Price isVirtual="true">
         hq://supplier3.com/PriceForUSA?ID=1234
      </Price>
   </Article>
   <Article ID="5">
      <ProductDescription>Window Regulator</ProductDescription>
      <Supplier>Supplier 4</Supplier>
      <Price isVirtual="true">
         hq://supplier4.com/SpecialPrice?SerialNo=WR4T
      </Price>
   </Article>
   ...
</Catalog>
```

Figure 2. Catalog of an Electronic Market Place Including Virtual Attributes (Shaded Gray)

object-specific parameters are introduced by "?" and constitute a "&"-separated key-value list. These object-specific parameters represent foreign keys on a virtual document at the data source where they are used to calculate the actual value of the virtual attribute. The object-specific parameters link together the static (à priori integrated) data portions at the market place and the dynamic portions obtained on demand from the data providers. All entries of the virtual attribute

with the same URI prefix must have the same object-specific parameter structure; each URI prefix leads to the instantiation of one sub-plan at the remote host. Each site can host multiple different sub-plans. Figure 2 shows a simple extension of a product catalog which is now presented in XML format. The virtual attribute *Price* of the first *Article* element indicates, that the price is calculated at the host `supplier1.com` using the sub-plan Electrical/Price for an object with key ProdID=CB1232.

2.3. Specification of HyperQueries

HyperQueries are the counterparts of virtual attributes and determine the dynamic parts of the mediated schema on demand. They are executed at the data source, may be arbitrarily complex, integrate applications and legacy systems, and may even involve user interaction. The most conventient way for stating HyperQueries is to use our SQL dialect where a HyperQuery accesses a virtual table called *HyperQueryInputStream*. This virtual table serves as the receiver of the input data objects that "flow through" the hyperlinks, i.e., the requests for the actual values referenced by hyperlinks. Within a HyperQuery only the object-specific parameters of the corresponding hyperlink and the application-specific parameters can be accessed. Additional attributes, e.g., customer-given attributes, of an input data object cannot be used within the HyperQuery; they are passed through. In our prototype system, alternatively, a HyperQuery can consist of arbitrarily complex Java operations which implement the iterator interface of [21].

Figure 3 shows the SQL formulations of two HyperQueries for the calculation of *Price*. The HyperQuery at Supplier 1 assumes a data source *Products* with attributes *ProdID* and *Price* and performs the join with the virtual table *HyperQueryInputStream* on the object-specific parameter of valid input hyperlinks. The HyperQuery at Supplier 4 is based on two local tables *Parts* and *BillOfMaterial*. Here, *Price* is calculated by summing up the prices of the components. Thereby, the HyperQuery invocations could be nested, if *p.Price* and *p.ComponentPrice* were again virtual attributes.

If an object is sent to a HyperQuery, the actual value is calculated from the object-specific parameters that are given by the URI and the application-specific parameters. The type of the actual value of the virtual attribute must coincide with the schema definition given at the market place; objects of incompatible type are discarded by the market place. If the type is single-valued and multiple values for the virtual attribute are computed, multiple values have to be returned.[2]

[2] For our example application of the car manufacturer's market place we assume that *Price* is a single-valued attribute.

```
select h.*, p.Price as Price
from Products p, HyperQueryInputStream h
where h.ProdID = p.ProdID
```

(a) *Electrical/Price* at Supplier 1

```
select h.*, sum(p.ComponentPrice) as Price
from Parts p, BillOfMaterial b, HyperQueryInputStream h
where h.SerialNo = b.SerialNo and b.ComposedOf = p.PartID
group by h.*
```

(b) *SpecialPrice* at Supplier 4

Figure 3. Two Example HyperQueries in Our SQL Dialect

If a HyperQuery is not able to compute the value of a virtual attribute the object is not returned. This reflects the semantics of a natural join and constitutes a simple mechanism of data cleaning (see [48] for more elaborate solutions). HyperQueries may initiate the instantiation of other HyperQueries when accessing virtual attributes. The results may flow back to the initiator of the HyperQuery where they are post-processed, or results are routed directly back to the client. We discuss both possibilities for the flow of results in more complex scenarios in the next section.

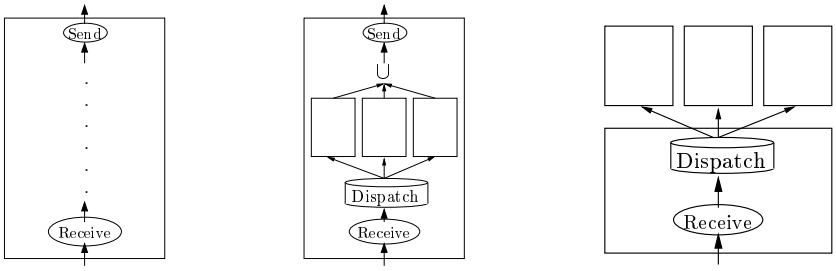
3. Executing HyperQueries

In contrast to other data integration approaches our HyperQueries reside at the data providers. In this section we present the execution of HyperQueries for the on demand data integration.

3.1. Templates for Sub-Plans

A HyperQuery that is stated as an SQL query is transformed into an operator tree using one of the three subsequently described templates depending on the complexity of the SQL query. The resulting operator tree is optimized using standard relational optimization methods and can be stored as a sub-plan at the data source. Figure 4 shows the three templates with the following characteristics:

Inner Sub-Plans Figure 4(a) shows sub-plans that have one input stream and one output stream. They are the simplest form of HyperQueries, where the *Receive* operator corresponds to the virtual table *HyperQueryInputStream* of the SQL query. The proper operator tree of the remaining query is built the standard way of compiling an SQL expression. The *Send* operator on top of the operator tree transmits the results of the HyperQuery execution back to the caller, e.g., the



(a) Inner Sub-Plans (b) Nesting Sub-Plans (c) Sequencing Sub-Plans

Figure 4. The Three Possible Templates for Sub-Plans

market place. These sub-plans are the innermost parts of the query execution where the actual values of virtual attributes are determined, e.g., a supplier reads the price information from a local database.

Nesting Sub-Plans As shown in Figure 4(b), these sub-plans contain a *Dispatch* operator that splits one input stream (which is provided by the *Receive* operator) into multiple output streams serving as input streams for the nested sub-plans. The *Dispatch* operator is the basic operator for processing HyperQueries (for implementation details see Section 5). The final *Union* operator (re-)merges the result streams of the nested sub-plans and produces one output stream. Thus, the flow of objects is totally encapsulated inside a sub-plan of this pattern. This pattern is a specialization of inner sub-plans and is used to build hierarchies of sub-plans. The client query including the HyperQuery execution is always transformed into a nesting sub-plan.

Sequencing Sub-Plans Sequencing sub-plans, as shown in Figure 4(c), contain the initial *Dispatch* operator that splits one input stream into multiple output streams; no final *Union* operator is given. The results of the dependent sub-plans are sent back to the *Union* operator of the surrounding sub-plan. Thus, objects that are once sent to a dependent sub-plan are never sent back to the initiating sub-plan and the data objects are beyond its control.

The main difference between nesting and sequencing sub-plans is the flow of results and the possibility of further processing of the virtual attributes after they have been materialized. Using nesting sub-plans the results flow from the HyperQuery execution back to the caller of the HyperQuery; whereas sequencing sub-plans send the results back to the surrounding sub-plan. This distinction is needed when executing multi-level HyperQueries which we present later in this section.

3.2. Processing Hyperlinks

The *Dispatch* operator splits one input stream into multiple output streams. This splitting is based on the virtual attribute and the objects are sent to the HyperQueries as determined by the embedded hyperlinks as follows:

1. The URI prefix is determined from the hyperlink.

2. If the referenced sub-plan has not yet been instantiated at the remote host, an instantiation request is sent. This request contains additional data that can be used at the remote host to parameterize the sub-plan. These "global parameters", e.g., the preferred currency of the client, stem from the context of the client[3] .

3. Once the sub-plan has been instantiated at the data source all objects with the same URI prefix of the hyperlink are sent there. Thereby, whole data objects, i.e, the hyperlinks and the additional elements of the input object, are transferred. This is usual in traditional database query processing and compliant with the iterator model, but different from hypertext processing, where only the request is sent and the resulting information is returned to the client. In Section 5 we discuss the splitting of objects and sending only the necessary parts.

3.3. Processing Simple Queries

We illustrate the incremental plan generation and execution of the query shown in Figure 1 using the database extension of Figure 2. Figure 5 shows a sample XML document *NeededProducts* that forms the "seed" of the HyperQuery invocation at the client.

For simplicity we substituted in Figure 6 the concrete data objects by symbols, where $\square$ and $\triangle$ denote the two battery objects, $\bigcirc$ denotes the tires object, and $\diamond$ denotes the spark plug object. Figure 6(a) shows the start of the query execution: The user-stated plan is instantiated with a scan of the *NeededProducts* document at the client. The attributes *Price* and *Supplier* of the *Catalog* document are joined (indicated by $\bowtie_{\smile}$) to the input objects. The vertical hatch indicates the joined objects.

The *Dispatch* operator splits the stream of objects into multiple output streams based on the virtual attribute *Price*. In Figure 6(b)

[3] In our market place "global parameters" are obtained from the registry information of the clients.

```
<NeededProducts>
   <Article>
      <ProductDescription>Battery, 12V 32A</ProductDescription>
      <OrderQuantity>500</OrderQuantity>
   </Article>
   <Article>
      <ProductDescription>Battery, 12V 55A</ProductDescription>
      <OrderQuantity>750</OrderQuantity>
   </Article>
   <Article>
      <ProductDescription>Tires 175/65TR14</ProductDescription>
      <OrderQuantity>1000</OrderQuantity>
   </Article>
   <Article>
      <ProductDescription>Spark Plug VX</ProductDescription>
      <OrderQuantity>8000</OrderQuantity>
   </Article>
</NeededProducts>
```

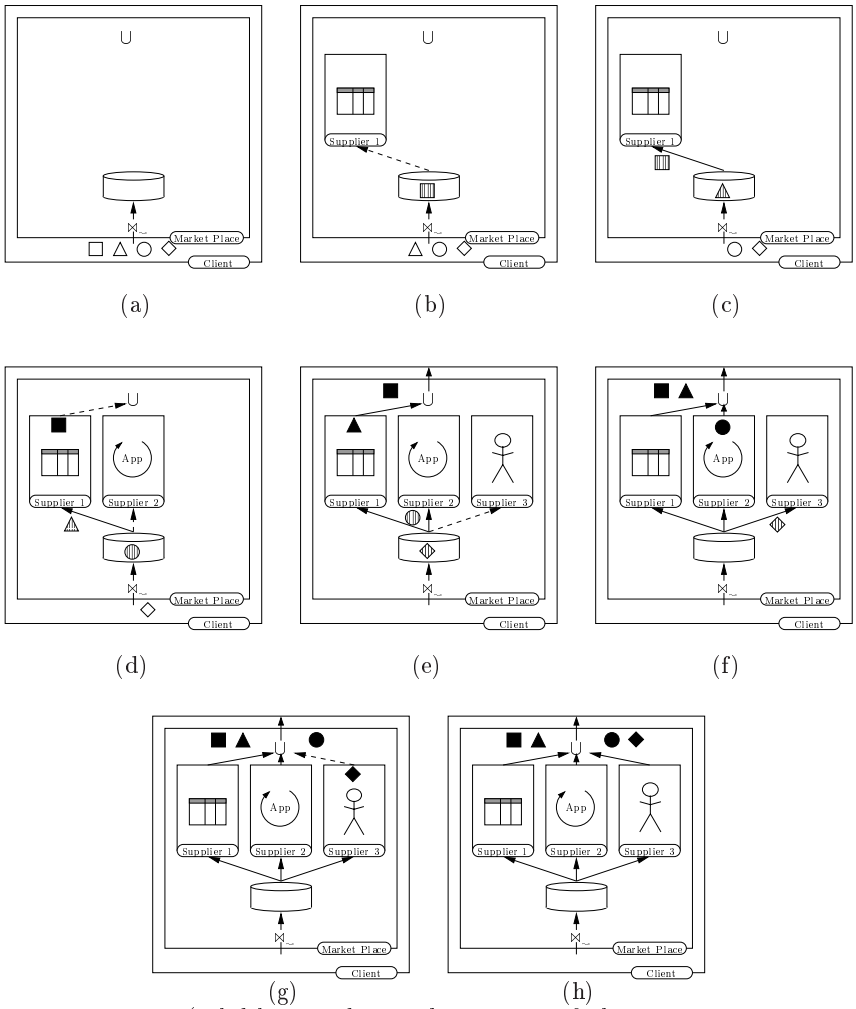
Figure 5. Some Needed Products of the Car Manufacturer

the first object for Supplier 1 passes the *Dispatch* operator which requests[4] the sub-plan Electrical/Price at Supplier 1. The *Dispatch* operator routes all objects referencing the same sub-plan to the same instance of the sub-plan (Figure 6(c)/(d)). Figure 6(d) also shows the processing of the ◍ object at the market place. The *Dispatch* operator sends an instantiation request to Supplier 2 where a complex "black-box" application (App) calculates the price. Concurrently, the *Price* has been added to the ▥ object at Supplier 1. The resulting ■ object[5] can be forwarded to the final *Union* where an input stream is requested.

After the registration of the new input stream at the *Union* the HyperQuery at Supplier 1 sends the ■ object back to the market place. The *Price* is inserted into the next data object ◮ (Figure 6(e)). The market place routes the ◍ object to Supplier 2 and requests the instantiation of the sub-plan *PriceForUSA* for the last input object ⧫ at Supplier 3 where a user enters the *Price* using a GUI. In Figure 6(f) the ⧫ object reaches Supplier 3, Supplier 2 has inserted the pricing information into its ◍ object and sends the resulting ● object to its target. Supplier 1 routes its ▲ object to the *Union*.

Supplier 2 sent the ● object to the *Union* and Supplier 3 inserted the *Price* for the ⧫ object (Figure 6(g)). The routing of the ◆ object leads to a request of an additional input stream at the *Union*. Figure 6(h) depicts the result, where the actual values of all input objects have been inserted and the resulting objects have reached the *Union*. Based on these data objects the query is processed further, i.e., the sorting by

[4] Note that all sub-plans are instantiated only once for a query.
[5] Objects with fully materialized virtual attributes are visualized in solid black.



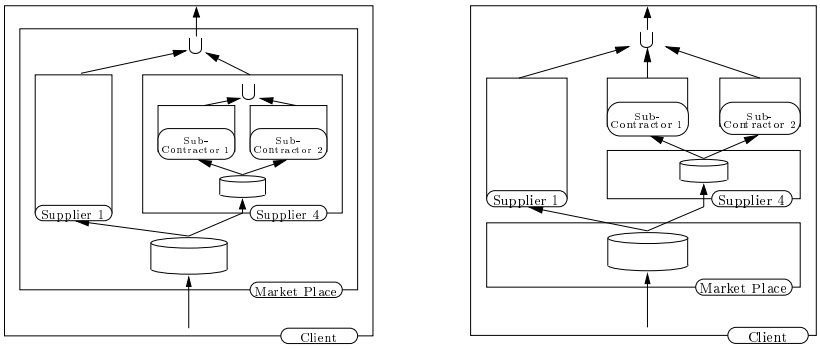
(solid lines indicate the routing of objects,
dashed lines indicate the instantiation of sub-plans)

Figure 6. Routing of Objects & Instantiation of Sub-Plans

ProductDescription and *Price* is done. After the complete execution of the query, all instantiated (intermediate) sub-plans and the user-stated query plan are closed and cleaned up in reverse order of instantiation.

3.4. Processing Multi-Level HyperQueries

So far we have demonstrated the instantiation of sub-plans for one-level HyperQuery processing. The HyperQuery concept is, as mentioned before, not restricted to one level. While processing a HyperQuery at a data source, other hyperlinks may be encountered which theirselves



(a) Hierarchical HyperQuery Execution (b) Broadcast HyperQuery Execution

Figure 7. Kinds of HyperQuery Execution

invoke dependent HyperQueries. This enables us to structure the data integration process and provide transparent access to a chain of data providers. Figure 7 illustrates more complex example applications utilizing the nesting and sequencing templates for sub-plans of Figure 4. We only show the query plans after all sub-plans have been instantiated and omit the example data objects. The main difference between the two techniques is the flow of results from the innermost HyperQueries back to the user. In hierarchical mode the results flow back to the sites where the HyperQuery requests were initiated. In broadcast mode the result objects are not collected at the invoking intermediate, but, rather, they are routed directly to the user.

3.4.1. *Hierarchical HyperQuery Execution*
If a remote host encounters a virtual attribute that is needed for the further execution of the HyperQuery, the remote host acts as intermediary and initiates a nested sub-query at another remote host using the pattern of Figure 4(b): The data objects flow from the surrounding sub-plan to the nested sub-plans, where the value of the virtual attribute is computed. Then the completed objects are sent back to the surrounding sub-plan, where they are processed further.

The nesting of sub-plans may be arbitrarily deep using again nesting or sequencing templates. In Figure 7(a) Supplier 4 executes the HyperQuery of Figure 3(b) and accesses the virtual attribute *ComponentPrice*. The HyperQuery of Supplier 4 has to be hierarchical, i.e., the nesting template for sub-plans has to be used, as the actual value of *ComponentPrice* is post-processed at Supplier 4. The virtual attributes at the levels of the nesting may differ, e.g., the outer virtual attribute could be *Price*, while the inner is *ComponentPrice*.

3.4.2. *Broadcast HyperQuery Execution*
If a hyperlink is encountered within a HyperQuery and the results need not be processed any further, the evaluation can be delegated to other HyperQueries. Using sequencing sub-plans data objects are forwarded to further sub-plans. It is the task of these sub-plans to determine the value of the virtual attribute and to send the resulting objects back to the initiator of the query. The prerequisite is that the virtual attributes are the same for both levels. The *Union* of the surrounding sub-plan merges the results of the inner sub-plans. In Figure 7(b) Supplier 4 has two subsidiary companies and delegates the incoming HyperQuery requests to Sub-Contractor 1 and 2 without post-processing the results.

The main advantage of broadcast processing is the quick forwarding of data objects without handling them again at the delegating site. The tradeoffs are (1) that the virtual attributes must coincide in all sequencing sub-plans and (2) that data from many unknown sources are provided to the collecting *Union* operator. In hierarchical mode the *Union* operator is informed from the *Dispatch* operator which data sources provide data. Nevertheless, as each site decides autonomously, which kind of sub-plan to executed, both hierarchical and broadcast execution of HyperQueries is possible within the execution of one query.

4. A Reference Architecture for Dynamic Market Places

In this section we propose a reference architecture for building scalable, distributed, electronic market places. Figure 8 gives an architectural overview of the whole market place federation, where three kinds of participants are distinguished: the market place, the customers, and the suppliers. The solid HQ annotated arrows reference HyperQueries at the data sources. The SOAP annotated arrow indicates that data integration is done via a Web service at the supplier outside the federation. These Web services are accessed via SOAP communication. We explain this extension of virtual attributes subsequently. Finally, the dashed lines represent the registration of new suppliers.

4.1. An Overview of the Reference Architecture

The main task of the market place is to act as an intermediary between participants. Therefore, the market place offers a mediated schema for integrating the suppliers' data sources to enable transparent access by the customers. The market place hosts the more static data, in particular, the product catalog and registry information, and executes queries accessing virtual attributes by distributing the requests to the participating suppliers.

The market place offers two basic interfaces: Customers pose their queries against the mediated schema using the *query interface*. The

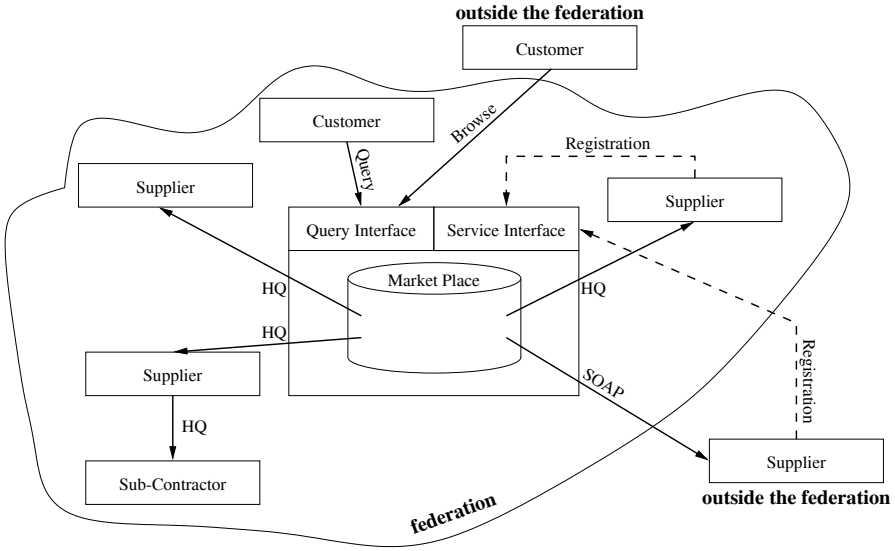
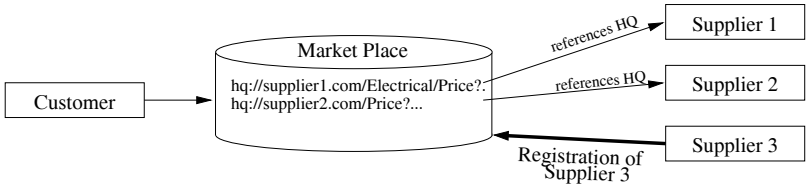


Figure 8. Architectural Overview of the Market Place Federation

service interface is used for all administrative tasks which are realized as Web services. The administrative tasks include registration of the static information at the market place. Thus, suppliers are able to manage both their registered products in the product catalog at the market place and provide meta information about the used HyperQueries for consistency checks. New customers and suppliers can autonomously join the market place as participants by submitting their registry information as XML documents to the market place. This makes the market place very scalable, as the administration of the data is done autonomously by the data providers.

Customers have to register at the market place using the service interface. Certificate-based authentication secures the registration process in which a customer profile, e.g., the location of the customer, the preferred currency, is transmitted. This profile is used for the global parameters during HyperQuery processing. The customers pose queries against the query interface of the market place. This can be done either by a browser accessing the web portal of the market place or by a client program. The client program is executed at the customer's site. It includes a full query processor and allows the seamless integration of data located at the customer into query processing. The benefit of the web portal is, that customers outside the federation do not need to install any additional software, whereas the fully integrated method requires the installation of the query processing system at the customer's site.

(a) Before: Execution of a Query



(b) Registration of a New Supplier

(c) Registration of a New Product

(d) After: Execution of a Query

Figure 9. A New Supplier Joins the Market Place

The suppliers can join and leave the market place at any time without affecting any other participants using the service interface. This means, that new data sources contribute data to the mediated schema at the market place and provide HyperQueries at their sites. Figure 9 illustrates the registration of a new supplier with products and the transparency of HyperQuery execution: At first, a customer poses a query involving data of Supplier 1 and Supplier 2. When a new supplier registers at the market place and uploads the static parts of the mediated data, virtual attributes are embedded for dynamic information. If a customer poses again the same query, the dynamic part of the data of Supplier 3 is also incorporated into the result. Figure 10 gives an example XML document for the registration of one product at the market place. Supplier 3 sends this XML document to the service interface of the market place. The suppliers have to provide HyperQueries

```
<ProductRegistration>
   <Article>
      <ProductDescription>Spark Plug VX</ProductDescription>
      <Supplier>Supplier 3</Supplier>
      <Price isVirtual="true">
         hq://supplier3.com/PriceForUSA?ID=1234
      </Price>
   </Article>
</ProductRegistration>
```

Figure 10. XML Document for the Product Registration of Supplier 3 Including Virtual Attributes (Shaded Gray)

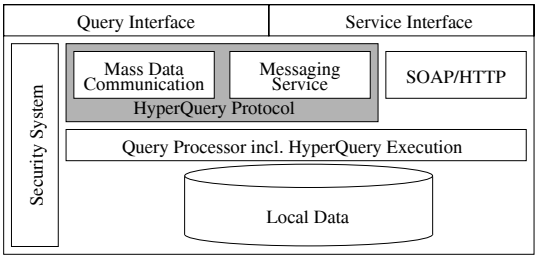


Figure 11. Details of one Participant of the Market Place Federation

at their (local) sites. As suppliers can be organized hierarchically, they can act as "mini-market places" and sub-contractors register at these suppliers. This enables us to structure the data integration process and multi-level HyperQuery processing is possible.

4.2. Detailed Description of Participants

We propose an open reference architecture for data integration systems and allow any host to participate in the federation. We distinguish two kinds of participants: participants that have installed the full system and participants using a SOAP connector to implement HyperQueries as Web services. Figure 11 shows details of a participant of the market place federation having installed the full system.

The major benefit of a supplier running the full system is the seamless integration of its sub-contractors into query processing. The main parts of the system can be characterized as follows: A database system stores all local data, e.g., at the market place the product catalog and the registry information. The query processor of the database system supports the execution of HyperQueries. Secure communication with remote hosts is ensured by a certificate-based security system which signs queries and requests when sending them to hosts and verifies them at the destination. The details of the security system are presented below. The communication layer of the system is divided into

two major parts: The HyperQuery protocol consists of a proprietary, efficiently working messaging service for the exchange of administrative control messages during HyperQuery processing as well as of mass data communication channels. The second part of the communication layer is SOAP/HTTP which provides openness for other systems to access the market place federation and vice versa. Thus, data integration is not restricted to the execution of HyperQueries, but Web services are also supported.

The application layer consists of the query interface and the service interface for the execution of Java-based Web services. WSDL [14] describes the Web services which implement the administration of the registered participants, static information, and meta data. The benefits are: On the one hand Web services can be composed and can seamlessly be integrated into more complex workflows at the participants. On the other hand new Web services can easily be integrated dynamically. Using this decentralized architecture the market place is very scalable, as the administration of the data is done locally at the data providers.

4.3. Implementing HyperQueries as Web Services

To enable any site to join the market place federation without installing the full system, HyperQueries can be realized as Web services. To access these Web services—which act as HyperQueries—hyperlinks to WSDL documents (instead of hyperlinks to HyperQueries) are embedded as virtual attributes into the mediated schema at the market place. WSDL [14] is the standard for describing the interface of Web services, i.e., the input and output parameters of Web services and the usage of protocols. Figure 12 shows a simplified WSDL document for the Web service *PriceService*. As one WSDL document may contain the specifications of multiple operations the hyperlinks to this document are extended with the name of the operation to be accessed within the WSDL document. The object-specific parameters are given in the hyperlink and define the values for the input parameters of the Web service. This way, Web services can be accessed as HyperQueries during query processing. The following hyperlink references the presented WSDL document:

```
http://www.supplier.com/PriceService.wsdl?
          operation_name=PriceClass1&
          Product=ExteriorMirror&Type=spheric&Color=blue
```

When encountering a hyperlink to a WSDL document the *Dispatch* operator executes the following procedure:

1. If not already loaded, the WSDL document is retrieved and cached.

```
<message name="getPriceRequest">
    <part name="Product" type="xs:string"/>
    <part name="Type" type="xs:string"/>
    <part name="Color" type="xs:string"/>
</message>
<message name="getPriceResponse">
    <part name="Price" type="xs:float"/>
</message>
<portType name="PriceService">
    <operation name="PriceClass1">
        <input message="getPriceRequest"/>
        <output message="getPriceResponse"/>
    </operation>
</portType>
```

Figure 12. Simplified WSDL Document of the Web Service PriceService (For simplicity the namespaces, WSDL header, bindings, etc. are omitted)

2. A proxy object for the Web service is generated from the WSDL document, i.e, a piece of Java code accessing the referenced Web service via SOAP is created, compiled and dynamically loaded into the runtime system. This proxy object is also cached for later reuse.

3. The object-specific parameters are bound dynamically to the proxy object.

4. A SOAP message is generated by the proxy object and is sent to the Web service.

The Web service executes the request and sends the results back to the market place. The market place passes the results to the *Union* operator of the corresponding query plan where, in return, further query processing proceeds. The *Dispatch* operator encapsulates the execution of the HyperQueries and the calls to Web services, so that the two techniques do not interfere and are possible within the same query execution.

4.4. Security Issues

Security is one of the crucial issues in an open and distributed query processing system. Users rely on the information provided by data integration systems, i.e., the stored and displayed information has to be secured against unauthorized access and malicious modifications. Especially in B2B market places which constitute the basis of economic activity, the security system has to cope with

- privacy, i.e., denying unauthorized sites access to sensitive data,
- integrity, i.e., denying unauthorized sites the modification of sensitive data,

- authentication, i.e., verifying the identity of a user,
- authorization, i.e., verifying, that a user has the permission to execute a requested sub-plan or an operation, and
- non-repudiation, i.e., no partner can deny the involvement in a transaction.

There have been many standardization efforts in the recent past on security in distributed environments, e.g., XML Signature [7] and XML Encryption [27] describe how to sign and encrypt XML documents. WS-Security [4] utilizes both standards to secure Web services by extending the header of SOAP messages with security relevant information. These approaches cover security issues of one single message exchange and are used within our proposed architecture, but the flow of data through multiple intermediaries requires extended security functionality.

Privacy and Integrity Communication streams between participants of the market place federation are protected by using the well-established secure communication standards SSL and/or TLS for encrypting and authenticating (digitally signing) messages. Both protocols can carry out the authentification of communication partners via X.509 certificates [26].

As business partners do not always communicate directly, but via intermediaries, e.g., market places, suppliers, or sub-contractors, using these secure protocols is not sufficient. The remote host that computes the actual value of the virtual attribute has to ensure that only the proper receiver has access to the data and intermediaries have to be withheld from reading it. Privacy and integrity of data can be ensured in multi-level HyperQuery execution by encryption and signatures.

Authentication In our market place framework authentication is used for three purposes: First, the customers, i.e., the clients of the data integration system, rely on true identity and the legally binding offers by the suppliers. Second, the suppliers have to know the identity of their customers due to special pricing conditions and contracting. Third, the carrier of the market place has to know who poses queries for its accounting. Each processed query incurs costs which have to be charged to the corresponding participants.

Authentication methods are based on passwords or X.509 certificates. We restrict the further discussion to certificates, i.e., each participant possesses an X.509 certificate. The certificate contains the public key that has been signed by an approved certification authority (CA). Digital signatures generated by the appropriate private key serve as a legal instrument for binding offers in e-commerce applications.

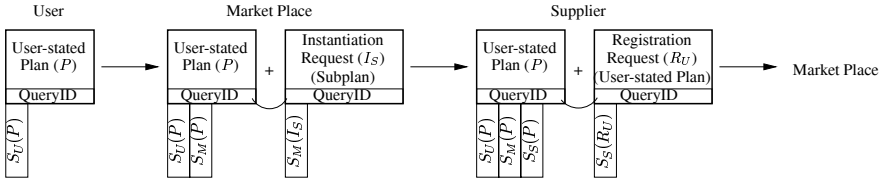


Figure 13. Chain of Signatures during HyperQuery Execution

In traditional distributed databases users sign query plans with their private keys and the authentication is performed at the participating providers. The signature of a query is verified and then the originator and the integrity of the query is verified reliably. This (simple) technique is not applicable to HyperQuery execution as the whole plan cannot be determined à priori and participants communicate through intermediaries. We devised the following extended authentication method for (multi-level) HyperQuery execution whereby we use the notation derived from [52] for the cryptographic functions assuming three communication partners:

1. user U who poses the initial query,

2. market place M which provides the mediated schema, receives the query of the user, executes a *Dispatch* operator on the virtual attributes, and sends instantiation requests to the data sources,

3. supplier S, i.e., a data source, which executes a HyperQuery.

Our approach hands over the initial plan and signs it multiple times. This constitutes a chain of trust. Figure 13 illustrates the algorithm with the instantiation of a sub-plan at supplier S:

1. The execution of a query involving HyperQueries starts at the user where the initial plan P containing a query identifier is signed with the user's private key.[6] The generated signature is appended to P, leading to $(P, S_U(P))$.

2. Each time a *Dispatch* operator sends an instantiation request I to a remote host both P and I are signed with the private key of the host that executes the *Dispatch* operator, generating $(P, S_U(P), S_M(P))$ and $(I, S_M(I))$. In multi-level HyperQuery execution multiple signatures are appended to the initial plan.

[6] Note that a message digest is generated from the initial plan using a hash function such as MD5. Then, this message digest is signed.

As the instantiation request also contains the query identifier of P both parts are marked to belong together and a separation by a malicious third party is detected by the receiver. A registration request at a *Union* is treated the same way. The receiver authenticates both the intermediary and the user by applying the following three steps:

1. The sending host is verified using SSL. Both received parts of the request, i.e., the initial plan and the instantiation request, have to be signed at last by the market place. Thus, the two verification tests are successful: $V_M(P, S_U(P), S_M(P))$ and $V_M(I, S_M(I))$.
2. Then, the first signature of the initial plan is checked with the public key of the user by $V_U(P, S_U(P))$.
3. The query identifier must coincide in the initial plan and the instantiation request.

On success of all three steps the supplier can reliably identify both the user and the market place. When the supplier sends back the results to the market place, a registration request R is sent to the *Union* operator at the market place, the supplier signs the request and the initial query and sends them both to the market place. At the market place the same authentication process is repeated with reversed roles.

Authorization The participants enforce their local authorization policy autonomously utilizing a role-based access control (RBAC) model [50] to specify authorization rules. RBAC distinguishes between users, roles which are assigned to users, and permissions which are assigned to roles. Each host establishes its own local rules, that declare which HyperQuery may be instantiated by which user. For instance, a supplier could have an exclusive contract upon a special kind of good with a certain customer. This can be implemented by a HyperQuery that is just used for this particular product group. The supplier stores the RBAC information that only the "exclusive" customer may execute this HyperQuery. On the instantiation of the HyperQuery the identity of the customer is checked and, if authorized, the HyperQuery is instantiated; otherwise the instantiation request is refused and an error is returned to the invoker of the HyperQuery.

Non-Repudiation It is essential for e-commerce applications that partners cannot deny the involvement in transactions. Following the traditional data warehouse approach this would be difficult, as the market place cannot legally sign the offers of the suppliers. Using HyperQueries, the market place acts as an intermediary and the offers are processed and legally signed by the suppliers themselves. As

it is currently not clear in some countries, whether an automatically processed and signed document is legally binding, we can use an operator for human input, that signs the produced objects on demand and explicitly expresses the declaration of intention.

5. Optimization and Implementation Details

In this section some optimization and implementation issues concerning the efficient and robust execution of HyperQueries are discussed.

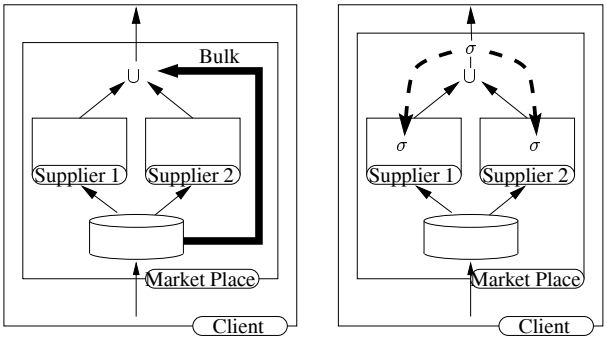
5.1. Bypassing of Attributes

The intermediary defines the mediated schema including meta information, e.g., which attributes are virtual. User queries include attributes which are necessary for query processing, such as virtual attributes, and bulk attributes, such as images or product descriptions. In the naïve approach whole objects consisting of both necessary and bulk attributes are sent to the HyperQuery. As the bulk attributes are not needed for the calculation of the virtual attribute at a remote host, they can be eliminated when passing the *Dispatch* operator. These attributes are sent to the final *Union* and are re-merged to the resulting objects after the actual value of the virtual attribute has been calculated. When the bulk attributes are eliminated a unique sequence number is added both to the bulk parts and the remaining data objects, i.e., the virtual attributes. Using these sequence numbers the bulk objects can be merged to the corresponding data objects at the *Union* operator.

This optimization method is similar to bulk bypassing [10, 15] in central databases. Figure 14(a) illustrates the bypassing of bulk objects around HyperQueries. Especially in multi-level HyperQuery execution this decreases the amount of transferred data and reduces the execution time in slow and bursty networks. This can also be applied when sensitive information has to be withheld from the remote hosts without expensive encryption of the data.

5.2. Predicate Migration

Predicates on virtual attributes cannot be evaluated before the actual values have been materialized. To reduce the amount of transferred data, the intermediary pushes these predicates from the initial query "into" the HyperQueries at the remote hosts so that only relevant data objects are returned. For example, adding the selection predicate $c.Price < 1500$ to the query of Figure 1 we show how predicate migration works: Without optimization the selection is applied late and the HyperQueries send all products back to the market place; pushing the selection down to the HyperQueries, less objects are transferred.



(a) Bulk Bypassing (b) Predicate Migration

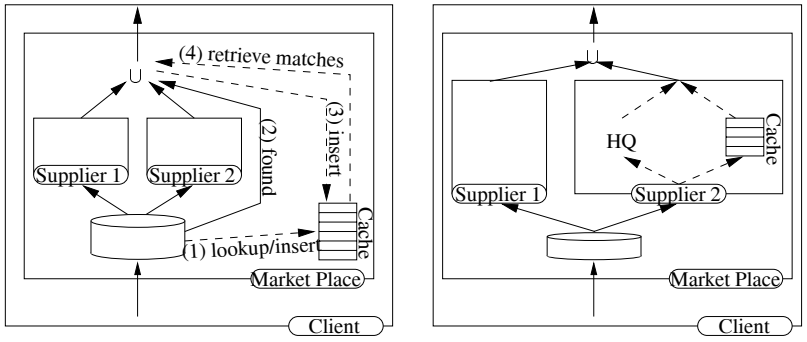
Figure 14. Illustrating Optimization Techniques (I)

The implementation of this optimization is straightforward: The selection predicate is sent to the remote site during the instantiation of the sub-plan. Before sending objects back to the market place a data provider performs the selection. Figure 14(b) illustrates the migration of predicates.

5.3. Caching at the Intermediary and at the Remote Hosts

Due to duplicates (which may be produced by preceding joins) the same virtual attributes may be evaluated multiple times. This can be avoided by caching.[7] Evaluating virtual attributes is similar to the invocation of expensive methods, but in contrast to [24] this is done asynchronously, i.e., objects are sent to sub-plans, before the results of previous objects are returned. Thus, it is not sufficient to store only the returned values. We also have to keep book of objects that were sent to sub-plans and have not yet produced a result. Figure 15(a) depicts the (hash table-based) caching of virtual attributes. On any input object the *Dispatch* operator probes the hash table (1). A cache hit is directly sent to the *Union*, bypassing the HyperQueries (2). Otherwise, the object is inserted into the hash table as a request. If this was the first request for this URI, the object is sent to the corresponding HyperQuery. If a result from a HyperQuery is received by the *Union*, it is inserted into the hash table (3) and the pending objects with the same URI are returned (4). If the results are highly dynamic and for coherence reasons cannot be re-used in another query, the hash table has to be discarded when the query execution has finished. If the remote hosts give TTLs, this approach can be extended to inter-query caching, where results are cached until expiry.

[7] The prerequisite for this is that multiple invocations of a HyperQuery with the same parameters return the same value.



(a) Caching at the Market Place (b) Caching at Supplier 2

Figure 15. Illustrating Optimization Techniques (II)

While this kind of caching takes place at the intermediary, the remote hosts cache requests and their results to reduce the processing overhead at their sites. Therefore, suppliers store for each requested URI the resulting object in a hash table and return the stored object on any further request of the same URI. This way, the execution of HyperQueries is shortcut which is especially in multi-level HyperQuery execution an effective optimization technique. Figure 15(b) shows this inter-query caching technique for Supplier 2 where the execution of the proper HyperQuery (HQ) is bypassed.

5.4. Multiple Virtual Attributes

If a query accesses multiple virtual attributes the naïve execution strategy would sequentially request at first the value of the first virtual attribute, then the value of the second virtual attribute, etc. If all virtual attributes of an object are evaluated at the same site, the requests can be bundled. The intermediary generates a plan that contains one *Dispatch* operator for *all* virtual attributes whose evaluation can be combined. During the execution the *Dispatch* operator sends the list of all requested virtual attributes with the instantiation request for one remote sub-plan. When an object passes the *Dispatch* operator, it is routed to the sub-plan, where the actual values of all virtual attributes are determined at once.

If virtual attributes, e.g., the price and the rating by an independent organization, are evaluated at the different sites, the calculation can be parallelized, anyway: The *Dispatch* operator instantiates multiple HyperQueries and sends *one* input object with a unique sequence number to *all* its corresponding sub-plans in parallel. The *Union* re-merges the resulting data objects of different HyperQueries using the sequence

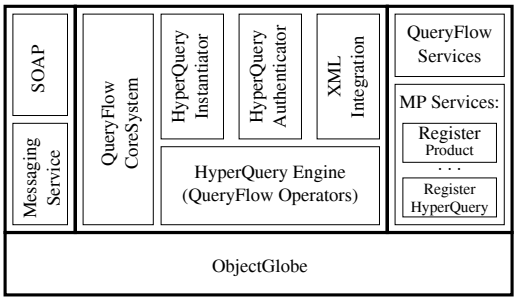


Figure 16. Architecture of the QueryFlow System

number. When the *Union* has re-merged all requests belonging together into one object, this result is passed to the next operator.

5.5. Our Prototype Implementation: The QueryFlow System

Having described the reference architecture of an open and scalable dynamic electronic market place in Section 4, we present our implemented QueryFlow system. We made an effort to rely on standards such as SQL, XML, XML Schema, X.509 certificates [26], XML Signature [7] and SOAP [57]. Figure 16 depicts the basic components:

- The query processing capabilities of QueryFlow are based on *ObjectGlobe*, a distributed and open query processor for data sources on the Internet. Braumandl et al. [11] give a full description of the ObjectGlobe system.

- The *HyperQuery Engine* combines all operators for HyperQuery processing, i.e., the *Dispatch* operator for resolving virtual attributes, and operators for the optimization of HyperQuery execution.

- The *QueryFlow CoreSystem* manages the instantiated HyperQueries and the distribution of one HyperQuery to multiple physical hosts including data structures such as caches and administrative data of the executed HyperQueries.

- The *HyperQuery Instantiator* manages the instantiation of HyperQueries at remote sites. The HyperQueries are stored in a hierarchically structured repository that can reside on top of the file system, a database, or a Web server.

- The certificate-based *HyperQuery Authenticator* signs requests and queries when sending them to hosts and verifies the authentication of incoming requests. This component is based on

the security infrastructure of ObjectGlobe and implements the presented security measures.

- The *XML Integration* component realizes the functionality to access XML data sources. We provide an interface for querying both relational and XML data sources using XQuery. These queries are transformed into SQL queries using methods proposed in [53, 54].
- Finally, we implemented some market place specific Web services for the administration of the market place, e.g., for registration of suppliers, products, and HyperQueries. These services can be accessed via SOAP or a proprietary messaging service.

5.5.1. *The Distributed Query Processor ObjectGlobe*
The idea of the underlying ObjectGlobe is to create an open market place for three kinds of suppliers: *data providers* supply data, *function providers* offer query operators to process data, and *cycle providers* are contracted to execute query operators. Of course, a single site can comprise all three services, i.e., act as data-, function-, and cycle-provider. ObjectGlobe enables applications to execute complex queries which involve the execution of operators from multiple function providers at different sites (cycle providers) and the retrieval of data from multiple data sources.

The whole system is written in Java for two reasons: First, Java is portable, so that the system can be installed with very little effort. In particular, hosts need to install the system and can very easily join a market place by inserting hyperlinks at the intermediary and providing the corresponding HyperQueries. Second, Java provides secure extensibility. Like the system itself, user-defined query operators are written in Java, they are loaded on demand (from function providers, e.g., the market place host or third-party vendors), and they are executed at the cycle providers in their own Java "sandbox" [45]. A user-defined, application-specific query operator must implement the *open*, *next*, *close*, and *reopen* methods following the iterator model [21].

5.5.2. *Operators for HyperQuery Processing*
All operators for HyperQuery processing, e.g., the asynchronous *Send* and *Receive* operators, the *Dispatch*, and the *Union* operator, are pipelined operators and therefore well suited for query processing on the Internet and processing streaming data. The *Dispatch* operator is a non-standard operator and splits one input stream into multiple output streams. As we want the concurrent and independent routing of objects to the instantiated HyperQueries, the *Dispatch* operator creates one thread for each output stream. All threads share one common input

stream, from which each one selects its relevant objects. The *Dispatch* operator coordinates the threads and keeps book of them.

5.5.3. *Data Sources for HyperQuery Execution*
The extensibility of the query processor is important, as each participant of the market place federation has several alternatives for implementing HyperQueries. Thus, query plans can be adapted to the companies' local systems. The query plans may contain different kinds of operators which can be characterized by the origin of the processed data:

Database Queries The simplest kind of HyperQueries are SQL queries as shown in Section 2. They are transformed into a tree containing physical operations, e.g., joins, selections, projections, and sorting. Dynamic loading of operators enables the administrator of the local host to integrate new and more efficient database operations into the query execution. One example of such a new database operation is a wrapper that accesses a relational database system using JDBC. This wrapper makes the integration of existing commercial DBMSs straightforward.

Applications If complex business applications, e.g., Enterprise Resource Planning systems like SAP R/3, spreadsheet analysis, etc., or legacy systems need to be accessed, wrappers for these applications have to be integrated into the query plan. This is done in the same way as database systems are integrated: the wrappers just have to obey the iterator interface. The applications are automatically invoked on any incoming data object and the actual value of the virtual attribute is calculated from the current input object. The connection of the QueryFlow system to legacy systems by wrappers means that data is only integrated on demand and the most coherent state of the data is returned.

Human Interaction HyperQueries may even incorporate human interaction where a user enters the value of a virtual attribute through a Java applet or a GUI. As these operators are executed at the data sources, sensitive data remains under full control of the data providers. These operators have two main parts: a server part, implementing the iterator interface, is specified in the query execution plan, and runs as a part of the query execution. The corresponding input interface acts as a client to this operator.

6. Performance Analysis

In this section we present benchmark results obtained from our QueryFlow system. In particular, we concentrate on investigating the

```
select PS_PARTKEY, PS_SUPPKEY, PS_COMMENT, '[ virtual_attrs ]'
from   PARTSUPP
where PS_PARTKEY < '[ sel ]'
```

(a) The User's Query

```
select h.*, p.P_RETAILPRICE as PS_SUPPLYCOST
from   HyperQueryInputStream h, PART@SUPP_i p
where h.PARTKEY = p.P_PARTKEY
```

(b) The HyperQuery at Supplier i

Figure 17. Queries used for Performance Investigation

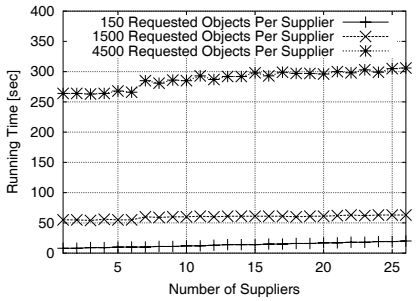
scalability of our approach in a distributed environment and show the effectiveness of the combination of multiple *Dispatch* operators.

6.1. Experimental Environment

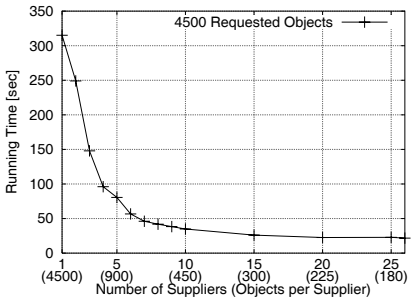
Our test scenario constitutes a market place with 26 suppliers where we adapted the well-known TPC-D [63] benchmark suite of scale factor 1.0. To suit our limited benchmark environment, we distributed the 10000 suppliers of TPC-D round robin by *S_SUPPKEY* to 26 hosts. The *PARTSUPP* table represented the product catalog at the market place and the *PART* table was partitioned horizontally to obtain several $PART@SUPP_i$ tables that contained the parts produced by Supplier i. Thus, each supplier offered approximately 30000 parts, whereby each part was produced by 4 suppliers which lead to 800000 entries at the market place and 200000 distinct parts. *PS_SUPPLYCOST* and *PS_AVAILQTY* became virtual attributes. The databases were stored in proprietary partitions on the file system. Each participant ran its database server on a separate host, whereby the market place was placed on a Sun Enterprise 450 with four 400 MHz UltraSparc II processors and 4 GByte memory. The machines were Sun Ultra 10 with 1 UltraSparc IIi processor at 333 MHz and 128 MByte memory. All hosts were in the same 100 MBit LAN, running Solaris 2.7 and using Sun's JDK 1.3. The security component was deactivated, as it was not our intent to measure the overhead for decryption, encryption, and authentication. Instead, our interest was on pure query processing performance. Within the HyperQueries we simulated the access to legacy systems or applications and slowed down their execution. Figure 17 shows the user's query and the HyperQuery executed at Supplier i.

6.2. Scalability of HyperQuery Processing

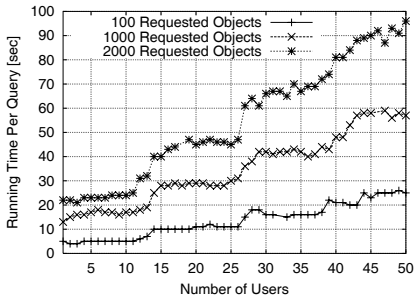
Measuring the scalability of our HyperQuery processing technique is divided into three parts. At first we determined the behavior of the system under a growing number of suppliers, then we varied the number of users. In both experiments we queried only one virtual attribute



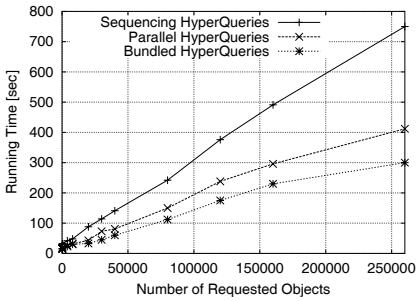
(a) Scaling the Number of Suppliers (Requesting from each Supplier 150/1500/4500 Objects)



(b) Scaling the Number of Suppliers (Adapting the Number of Requested Objects)



(c) Scaling the Number of Users (Varying the Number of Users from 1..50)



(d) Accessing Multiple Virtual Attributes (Scaling the Number of Requested Objects)

Figure 18. Investigating the Scalability

PS_SUPPLYCOST. Finally, we show the performance gain of parallel evaluation of multiple virtual attributes.

6.2.1. *Scaling the Number of Suppliers*
In the first set of benchmarks we determined the behavior of the system under a growing number of suppliers.

Requesting from each Supplier 150/1500/4500 Objects We varied the number of requested suppliers from 1 to 26 and requested from each supplier 150/1500/4500 objects. Thus, we got an increasing number of result objects. Figure 18(a) shows the running times which are almost constant within one test set. Thus, the overall performance is determined by the execution time of the HyperQueries at the remote hosts.

Adapting the Number of Requested Objects In the previous experiment the number of returned objects was increased with the number of re-

quested suppliers. Now we adapted the number of requests to obtain 4500 objects in total and measured the parallelization of the requests from 1 to 26 suppliers. Figure 18(b) shows that the total running time decreases with more suppliers because of the parallelization effects at the market place.

6.2.2. *Scaling the Number of Users*
In this test we demonstrate the scalability of the market place when varying the number of users that simultaneously posed queries at the market place. Each of the users requested 100/1000/2000 products from two suppliers each. This leads to 13 user queries that can be run without accessing one of our 26 suppliers multiple times. We varied the number of querying users from 1 up to 50. Figure 18(c) gives the running times which are almost constant within the segments from 1 to 13, 14 to 26, 27 to 39, 40 to 50 users and jump up in step function manner. These steps are caused by multiple accesses of the same suppliers.

6.2.3. *Evaluating Multiple Virtual Attributes*
At last, we demonstrate the benefits of bundling requests for multiple virtual attributes incorporating all 26 suppliers. The user's query accessed the two virtual attributes *PS_SUPPLYCOST* and *PS_AVAILQTY*. We varied the selectivity of the query, requesting from 100 up to 260000 data objects. Figure 18(d) shows the running times for the alternative execution plans. It can be seen that all execution times increase linear with the number of requested objects. This is not surprising as each supplier has a proportional ratio of the requested objects. The naïve plan (*Sequencing HyperQueries*) had two sequencing *Dispatch* operators requesting at first *PS_SUPPLYCOST* and then *PS_AVAILQTY*. The first optimized variant (*Parallel HyperQueries*) parallelizes the evaluation of both virtual attributes and nearly halves the running times as multiple round trips of objects are avoided. The second optimization variant (*Bundled HyperQueries*) combines the evaluation of both virtual attributes in one request and draws additional profit: the overhead of re-merging the duplicates of one input object at the *Union* leaves out.

7. Summary and Future Work

Electronic market places and virtual enterprises have become very important applications for distributed query processing. Building a scalable virtual Business-to-Business market place with hundreds or thousands of participating suppliers requires highly flexible and scalable distributed query processing capabilities. Architecting an electronic market place as a data warehouse-like approach by integrating *all* the data from *all* participating enterprises in one centralized repository

incurs severe problems. We first presented a new framework for dynamic distributed query processing based on so-called *HyperQueries* which are essentially query evaluation plans "sitting behind" hyperlinks. Our approach facilitates the pre-materialization of static data at the market place whereas the dynamic data remains at the data sources. In contrast to traditional data integration systems, our approach executes essential (dynamic) parts of the data-integrating views at the data sources. The other, more static parts of the data are integrated à priori at the central portal, e.g., the market place. The portal serves as an intermediary between clients and data providers which execute their sub-queries referenced via hyperlinks. We illustrate the flexibility of this distributed query processing architecture in the context of B2B electronic market places with an example derived from the car manufacturing industry.

Based on these HyperQueries, we proposed a reference architecture for building scalable and dynamic electronic market places. All administrative tasks in such a distributed B2B market place are modeled as Web services and are initiated decentrally by the participants. Thus, sensitive data remains under the full control of the data providers. We described optimization and implementation issues to obtain an efficient and highly flexible data integration platform for electronic market places. All proposed techniques have been fully implemented in our QueryFlow prototype system which served as platform for our performance evaluation.

In future work we will extend these query processing concepts to Peer-to-Peer (P2P) environments. In a recent paper [12] we describe a distributed index structure to efficiently find data in P2P networks, expand query plans on the fly, and place operators nearby the data sources. In this context, HyperQueries support the implementation of the index structures and the dynamic plan generation. This environment offers a wide field of novel aspects such as the clustering of data, the placement of operators, and load balancing.

Acknowledgements

We would like to thank Christina Popp, Peter Winklhofer, David Schmitz, and Stefan Brandl for their help in the implementation and the editor and the anonymous reviewers for their helpful comments.

References

1. Abiteboul, S., B. Amann, J. Baumgarten, O. Benjelloun, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo, 'Active XML Project'. `http://www-rocq.inria.fr/gemo/Gemo/Projects/axml/index.html`.

2. Abiteboul, S., A. Bonifati, G. Cobena, I. Manolescu, and T. Milo: 2003, 'Dynamic XML Documents with Distribution and Replication.'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. San Diego, CA, USA, pp. 527–538.
3. Arens, Y., C. A. Knoblock, and W.-M. Shen: 1996, 'Query Reformulation for Dynamic Information Integration'. *Journal of Intelligent Information Systems - Special Issue on Intelligent Information Integration* **6**(2/3), 99–130.
4. Atkinson, B., G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, C. Kaler, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon: 2002, 'Web Service Security (WS-Security)'. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp`.
5. Bakos, Y.: 1991, 'A Strategic Analysis of Electronic Marketplaces'. *MIS Quarterly* **15**(3), 295–310.
6. Bakos, Y.: 1998, 'The Emerging Role of Electronic Marketplaces on the Internet'. *Communications of the ACM* **41**(8), 35–42.
7. Bartel, M., J. Boyer, B. Fox, B. LaMacchia, and E. Simon: 2002, 'XML Signature'. `http://www.w3.org/TR/xmldsig-core/`. W3C Recommendation.
8. BEA, 'BEA WebLogic Enterprise Platform'. `http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/solutions/te%chnical/platform`.
9. Bernstein, P., N. Goodman, E. Wong, C. Reeve, and J. Rothnie: 1981, 'Query Processing in a System for Distributed Databases (SDD-1)'. *ACM Transactions on Database Systems* **6**(4), 602–625.
10. Braumandl, R., J. Claussen, A. Kemper, and D. Kossmann: 2000, 'Functional Join Processing'. *The VLDB Journal* **8**(3-4), 156–177. Invited Contribution to the Special Issue "Best of VLDB 98".
11. Braumandl, R., M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker: 2001, 'ObjectGlobe: Ubiquitous Query Processing on the Internet'. *The VLDB Journal: Special Issue on E-Services* **10**(3), 48–71.
12. Brunkhorst, I., H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner: 2003, 'Distributed Queries and Query Optimization in Schema-Based Systems'. In: *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*.
13. Casati, F., U. Dayal, and M.-C. Shan: 2001, 'E-Business Application for Supply Chain Management: Challenges and Solutions'. In: *Proceedings of the IEEE Conference on Data Engineering*. Heidelberg, Germany, pp. 71–78.
14. Christensen, E., F. Curbera, G. Meredith, and S. Weerawarana: 2001, 'Web Services Description Language (WSDL) 1.1'. `http://www.w3.org/TR/wsdl`. W3C Note.
15. Claussen, J., A. Kemper, D. Kossmann, and C. Wiesner: 2000, 'Exploiting Early Sorting and Early Partitioning for Decision Support Query Processing'. *The VLDB Journal* **9**(3), 190–213. Invited Contribution to the Special Issue "Best of VLDB 99".
16. Covisint: 2000. `http://www.covisint.com`.
17. Crescenzi, V., G. Mecca, and P. Merialdo: 2001, 'RoadRunner: Towards Automatic Data Extraction from Large Web Sites'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Rome, Italy, pp. 109–118.
18. Draper, D., A. Y. Halevy, and D. S. Weld: 2001, 'The Nimble XML Data Integration System'. In: *Proceedings of the IEEE Conference on Data Engineering*. Heidelberg, Germany, pp. 155–160.

19. Florescu, D. and D. Kossmann: 2001, 'An XML Programming Language for Web Service Specification and Composition'. *IEEE Data Engineering Bulletin* **24**(2), 48–56.
20. Goldman, R. and J. Widom: 2000, 'WSQ/DSQ: A Practical Approach for Combined Querying of Databases and the Web'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Dallas, TX, USA, pp. 285–296.
21. Graefe, G.: 1993, 'Query Evaluation Techniques for Large Databases'. *ACM Computing Surveys* **25**(2), 73–170.
22. Haas, L., D. Kossmann, E. Wimmers, and J. Yang: 1997, 'Optimizing Queries Across Diverse Data Sources'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Athens, Greece, pp. 276–285.
23. Halevy, A. Y., Z. G. Ives, P. Mork, and I. Tatarinov: 2003, 'Piazza: Data Management Infrastructure for Semantic Web Applications'. In: *International World Wide Web Conference*. Budapest, Hungary.
24. Hellerstein, J. and J. Naughton: 1996, 'Query Execution Strategies for Caching Expensive Methods'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Montreal, Canada, pp. 423–434.
25. Hellerstein, J. M., M. Stonebraker, and R. Caccia: 1999, 'Independent, Open Enterprise Data Integration'. *IEEE Data Engeneering Bulletin* **22**(1), 43–49.
26. Housley, R., W. Ford, W. Polk, and D. Solo: 1999, 'Internet X.509 Public Key Infrastructure Certificate and CRL Profile'. `http://www.rfc-editor.org/rfc/rfc2459.txt`.
27. Imamura, T., B. Dillaway, and E. Simon: 2002, 'XML Encryption Syntax and Processing'. `http://www.w3.org/TR/xmlenc-core`. W3C Recommendation.
28. Ives, Z., D. Florescu, M. Friedman, A. Levy, and D. Weld: 1999, 'An Adaptive Query Execution Engine for Data Integration'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Philadelphia, PA, USA, pp. 299–310.
29. Jhingran, A.: 2000, 'Moving up the food chain: Supporting E-Commerce Applications on Databases'. *ACM SIGMOD Record* **29**(4), 50–54.
30. Josifovski, V., P. Schwarz, L. Haas, and E. Lin: 2002, 'Garlic: A New Flavor of Federated Query Processing for DB2'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Madison, WI, USA, pp. 524–532.
31. Jr., R. J. B., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk: 1997, 'InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Tucson, AZ, USA, pp. 195–206.
32. Keidl, M., S. Seltzsam, K. Stocker, and A. Kemper: 2002, 'ServiceGlobe: Distributing E-Services across the Internet (Demonstration)'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. HongKong, China, pp. 1047–1050.
33. Keller, T., G. Graefe, and D. Maier: 1991, 'Efficient Assembly of Complex Objects'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Denver, CO, USA, pp. 148–158.
34. Kemper, A. and C. Wiesner: 2001, 'HyperQueries: Dynamic Distributed Query Processing on the Internet'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Rome, Italy, pp. 551–560.
35. Knoblock, C. A., S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada: 2001, 'The Ariadne Approach to Web-Based Information Inte-

gration'. *International Journal of Cooperative Information Systems* **10**(1-2), 145–169.
36. Kossmann, D.: 2000, 'The State of the Art in Distributed Query Processing'. *ACM Computing Surveys* **32**(4), 422–469.
37. Lenzerini, M.: 2002, 'Data Integration: A Theoretical Perspective'. In: *Proceedings of the ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*. Madison, WI, pp. 233–246.
38. Levy, A.: 2001, 'Answering Queries Using Views: A Survey'. *The VLDB Journal* **10**(4), 270–294.
39. Levy, A., A. Rajaraman, and J. Ordille: 1996, 'Querying Heterogeneous Information Sources Using Source Descriptions'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Bombay, India, pp. 251–262.
40. Levy, A. Y., D. Srivastava, and T. Kirk: 1995, 'Data Model and Query Evaluation in Global Information Systems'. *Journal of Intelligent Information Systems (JIIS)* **5**(2), 121–143.
41. Mena, E., V. Kashyap, A. P. Sheth, and A. Illarramendi: 1996, 'OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies'. In: *Conference on Cooperative Information Systems*. Brussels, Belgium, pp. 14–25.
42. Nejdl, W., M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser: 2003, 'Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks'. In: *International World Wide Web Conference*. Budapest, Hungary.
43. NET: 2001, 'Microsoft .NET'. `http://www.microsoft.com/net`.
44. Nodine, M. H., J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, and A. Unruh: 2000, 'Active Information Gathering in InfoSleuth'. *International Journal of Cooperative Information Systems* **9**(1-2), 3–28.
45. Oaks, S.: 1998, *Java Security*. Sebastopol, CA, USA: O'Reilly & Associates.
46. Papakonstantinou, Y., S. Abiteboul, and H. Garcia-Molina: 1996, 'Object Fusion in Mediator Systems'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Bombay, India, pp. 413–424.
47. Papakonstantinou, Y., A. Gupta, H. Garcia-Molina, and J. Ullman: 1995, 'A Query Translation Scheme for Rapid Implementation of Wrappers'. In: *Proceedings of the Conference on Deductive and Object-Oriented Databases (DOOD)*. Singapore, Singapore, pp. 161–186.
48. Rahm, E. and H.-H. Do: 2000, 'Data Cleaning: Problems and Current Approaches'. *IEEE Bulletin of the Technical Committee on Data Engineering* **23**(4), 3–13.
49. Rodriguez-Martinez, M. and N. Roussopoulos: 2000, 'MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Dallas, TX, USA, pp. 213–224.
50. Sandhu, R. S., E. J. Coyne, H. L. Feinstein, and C. E. Youman: 1996, 'Role-Based Access Control Models'. *IEEE Computer* **29**(2), 38–47.
51. SAP: 1999, 'Business Networking in the Internet Age'. Technical report, SAP White Paper. `http://www.sap-ag.de/germany/products/mysap/pdf/bus\_networking.pdf`.
52. Schneider, B.: 1996, *Applied Cryptography, Second Edition*. Chichester, UK: John Wiley & Sons.

53. Shanmugasundaram, J., J. Kiernan, E. J. Shekita, C. Fan, and J. Funderburk: 2001a, 'Querying XML Views of Relational Data'. In: *Proceedings of the Conference on Very Large Data Bases (VLDB)*. Rome, Italy, pp. 261–270.
54. Shanmugasundaram, J., E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald: 2001b, 'Efficiently publishing relational data as XML documents'. *The VLDB Journal* **10**(2-3), 133–154.
55. Shekita, E. and M. Carey: 1990, 'A Performance Evaluation of Pointer-Based Joins'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Atlantic City, NJ, USA, pp. 300–311.
56. Sheth, A. and J. Larson: 1990, 'Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases'. *ACM Computing Surveys* **22**(3), 183–236.
57. SOAP: 2003, 'Simple Object Access Protocol (SOAP) 1.2'. `http://www.w3.org/TR/soap12-part0`. W3C Recommendation.
58. Stonebraker, M.: 1985, 'The Design and Implementation of Distributed INGRES'. In: *The INGRES Papers: Anatomy of a Relational Database System*. Reading, MA, USA: Addison-Wesley.
59. Stonebraker, M., E. Anderson, E. Hanson, and B. Rubenstein: 1984, 'QUEL as a Data Type'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Boston, MA, pp. 208–214.
60. Stonebraker, M., P. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu: 1996, 'Mariposa: A Wide-Area Distributed Database System'. *The VLDB Journal* **5**(1), 48–63.
61. SunONE, 'Sun Open Net Environment (Sun ONE)'. `http://www.sun.com/sunone`.
62. Tomasic, A., L. Raschid, and P. Valduriez: 1996, 'Scaling Heterogeneous Databases and the Design of DISCO'. In: *Proceedings of the International Conference on Distributed Computing Systems*. Hong Kong, pp. 449–457.
63. TPC, T. P. P. C.: 1999, 'TPC Benchmark D (Decision Support)'. Standard Specification 2.1, Transaction Processing Performance Council (TPC). `http://www.tpc.org`.
64. WebSphere, 'IBM WebSphere'. `http://www.ibm.com/websphere`.
65. Wiederhold, G.: 1993, 'Intelligent Integration of Information'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Washington, DC, USA, pp. 434–437.
66. Wiesner, C., P. Winklhofer, and A. Kemper: 2002, 'Building Dynamic Market Places using HyperQueries'. In: *Proceedings of the International Conference on Extending Database Technology (EDBT)*. Prague, Czech Republic, pp. 749–752.
67. Williams, R., D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, and R. Yost: 1981, 'R*: An Overview of the Architecture'. IBM Research, San Jose, CA, RJ3325. Reprinted in: M. Stonebraker, Readings in Database Systems, Morgan Kaufmann Publishers, 1994, pages 515–536.
68. Xyleme, L.: 2001, 'A Dynamic Warehouse for XML Data of the Web'. In: *IEEE Data Engineering Bulletin, Vol. 24, No. 2*. pp. 40–47.
69. Yang, J. and M. P. Papazoglou: 2000, 'Interoperation Support for Electronic Commerce'. *Communications of the ACM* **43**(6), 39–47.
70. Zadorozhny, V., L. Raschid, M. E. Vidal, T. Urhan, and L. Bright: 2002, 'Efficient Evaluation of Queries in a Mediator for WebSources'. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*. Madison, WI, USA, pp. 85–96.