From Personal Desktops to Personal Dataspaces: A Report on Building the iMeMex Personal Dataspace Management System*

Jens-Peter Dittrich Lukas Blunschi Markus Färber
Olivier René Girard Shant Kirakos Karakashian Marcos Antonio Vaz Salles
ETH Zurich, Switzerland
dbis.ethz.ch | iMeMex.org

Abstract: We propose a new system that is able to handle the entire Personal Dataspace of a user. A Personal Dataspace includes all data pertaining to a user on all his disks and on remote servers such as network drives, email and web servers. This data is represented by a heterogeneous mix of files, emails, bookmarks, music, pictures, calendar data, personal information streams and so on. State-of-the-art tools such as desktop search engines and desktop operating systems (including the upcoming Vista) are not enough as they neither solve the problem of physical personal information independence (where is my data) nor format and data model independence (how is it stored and which application do I have to use in order to access that data). Our work builds on the visions presented in [DSKB05], which calls for a single system to manage the personal information jungle, and [FHM05], which advocates dataspaces as a new abstraction for information management. In contrast to [FHM05] this paper presents a concrete implementation of a Personal DataSpace Management System (PDSMS) termed iMeMex: integrated memex. We discuss the core architecture of iMeMex and services offered by our system. As we will show, a PDSMS can be seen as a system that occupies the middleground between a search engine, a database management system, and a traditional information integration system. A PDSMS has to bridge these separate worlds and requires: (1) no full control on data, i.e., data may be accessed bypassing the interfaces of a PDSMS, (2) simple keyword search on all data available in a dataspace without performing any semantic data integration, (3) rich querying able to mix structural, attribute, and content predicates, (4) pay-as-you-go integration capabilities, (5) the ability to define arbitrary logical views on all data, (6) durability and consistency guarantees to avoid loss of data assigned to a dataspace, and (7) update capabilities. iMeMex is the first implementation of a PDSMS we are aware of. This paper presents the architecture of iMeMex and reports on the current state of the iMeMex research project at ETH Zurich.

*This work is partially supported by the Swiss National Science Foundation (SNF) under contract 200021-112115.

1 Introduction

Dataspaces have recently been identified as a new agenda for information management [FHM05, HFM06, Mai06, HRO06]. In an nutshell, a *dataspace management system* is a new kind of information managing architecture that allows to manage *all* data pertaining to a certain organization, task, or person. In sharp contrast to existing information integration architectures a dataspace management system is a *data-coexistence approach*: it does *not* require semantic integration investments before services on the data are provided. For example, keyword searches should be supported at any time on all data (*schema later* or *schema never*). The existing dataspace can then be gradually enhanced by defining semantic connections between different components of a dataspace in a pay-as-you-go fashion [FHM05]. An important use-case of dataspaces are personal dataspaces, i.e., all electronic items that belong to a single person.

This paper aims to give an overview on the `iMeMex` project which was started two years ago at ETH Zurich and strives to build the first publicly available Personal DataSpace Management System (PDSMS). The current version of our software was shown in a demo at CIDR 2007 [BDG+07]. `iMeMex` is supported by the Swiss National Science Foundation (SNF) and includes a senior researcher, two Ph.D. students, currently three M.Sc. students as well as several short-term project work students (fourteen have completed their work so far). `iMeMex` is open source since December 15, 2006 (Apache 2.0 License) and a first developer version of the `iMeMex` server is available from our web-site `http://www.imemex.org`.

1.1 Background

Personal Information Jungle. In 1945, Bush [Bus45] presented a vision of a personal information management (PIM) system named *memex*. That vision has deeply influenced several progresses in computing. Part of that vision led to the development of the *Personal Computer* in the 1980ies. It also led to the development of *hypertext* and the *World Wide Web* in the 1990ies. Since then, several projects have attempted to implement other memex-like functionalities [FG96, Bel05, CRDS06, KBH+05]. In addition, PIM regained interest in the database research community [HED+03, DH05a]. Moreover, it was identified as an important topic in the Lowell Report [AAB+03], discussed in a VLDB panel [KWF+03], and became topic of both SIGMOD 2005 keynotes [Bel05, Mit05]. Furthermore, it was discussed in an NSF-sponsored workshop [JB05] and debated in a SIGIR 2006 PIM workshop [PIM].

Even though considerable progress has been made in the PIM area over the last decades, we argue that a satisfactory solution has still not yet been brought forward to many central issues related to PIM. This is illustrated in the following.

1.2 The iMeMex Vision

PIM today: Assume that Mr. John Average owns a set of devices including a laptop, a desktop, a cellular, and a digital camera. His personal files are spread out among those de-

vices and include music, pictures, pdf files, emails, and office documents. Today, Mr. Average has to copy files from one device to the other, he has to download data to his desktop to see the pictures he shot, he has to upload pictures to sites like *flickr.com* or *picasa.com* to share them with his family. He has to make sure to regularly backup data from the different devices in order to be able to retrieve them in case of a device failure. Further, he uses two different modes of searching: a local desktop search engine enabling search on his local devices and a web search engine enabling search on public web-sites. Mr. Average may organize his files by placing them in folder hierarchies. However, the files and data items stored on his different devices are not related to each other.

`iMeMex` vision of 2010: Mr. John Average still owns several nifty devices with growing processing and storage capabilities. Instead of handling 'devices' he assigns all his data to a logical *dataspace* named `John's space`. Whenever he listens to a piece of music, takes a picture, gets an email, etc., those items are assigned to `John's space`. His dataspace management system takes care of the low-level issues including replicating data among devices, enabling search and querying across devices. Whenever John Average wants to share data, he simply creates a subdataspace like `John's space:pictures` and selects a list of people who may see that data, e.g., his family or friends. There is no need to 'upload' or 'download' data: John's family and friends will just *see* John's pictures without requiring to access web-servers or messing around with files. The boundary between the Web and the different operating systems running on his local devices is gone. However, John Average still *owns* his data: all master copies of his data are physically stored on devices he owns. Searching his dataspace is not restricted to certain devices (like the local desktop), but includes all devices containing data assigned to his dataspace. Other than simple keyword queries, structural queries similar to NEXI [TS04] are enabled. John Average may also search and query the dataspaces of his friends and his family. The search granularity is fine-granular 'resource views' [DS06] and *not* files. Other than just searching or querying, John Average may also use `iMeMex` to integrate the information available in his dataspace or his friends' dataspaces in a pay-as-you-go fashion. Therefore, his dataspace management system analyzes the data and proposes relationships among data items. It enhances his dataspace over time and helps to turn a set of unrelated data items into integrated information. Finally, Mr. Average may also update data using `iMeMex`. However, he may still update his data using any of his applications, bypassing `iMeMex`.

2 Related Work

Figure 1 displays the design space of existing solutions for information management. The horizontal axis displays requirements for semantic integration, while the vertical axis, in contrast to [FHM05], displays the degree of update guarantees provided by different systems.

On the lower left corner of the design space we find **DBMSs**, which require high semantic integration efforts (upfront investment for schemas), but provide strong update guarantees (ACID). Examples of systems that attempted to apply DBMS technology to personal information include MyLifeBits [Bel05] as well as Rufus [SLS+93]. However, these solutions

incur high costs for semantic integration and require full control of the data. In sharp contrast to that, a PDSMS does not require schema-first modeling.

In the upper right of Figure 1, strictly opposed to DBMSs, we find **Desktop Search Engines (DSE)**. They do neither require semantic integration, nor full control of the data. The downside, however, is that a DSE does neither provide any update guarantees nor does it allow to include complex structural constraints, e.g., queries like `//mail//*[from="Jens Dittrich"]`. Moreover a DSE does not provide any semantic information integration capabilities. Examples of DSEs are Google Desktop [GDS], Apple Spotlight [AMS], Beagle [Bea], and Phlat [CRDS06].

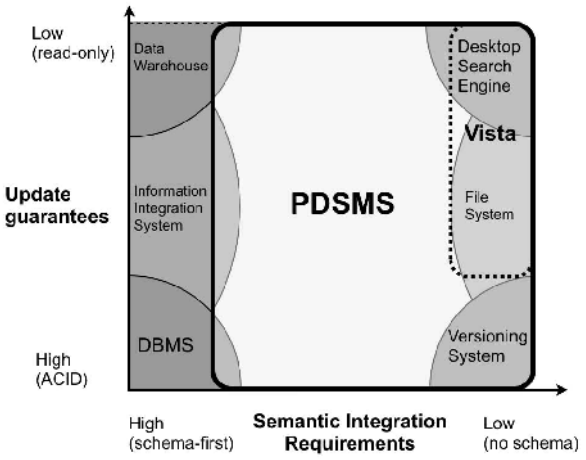

Figure 1: Design space of state-of-the-art information management systems: PDSMSs fill the space between existing specialized systems.

Figure 1 also shows **Traditional Information Integration Systems** in the middle-left: these systems require high semantic integration investments and vary in terms of their update guarantees. Several traditional information integration systems have used the relational model to provide a global, data source independent view of data [LRO96, ACHK94]. The Information Manifold [LRO96], for example, is based on the idea of a global schema on top of which the data sources may be expressed as relational views (LAV). Semistructured approaches, first introduced by TSIMMIS [PGMW95], have been proposed as an alternative to self-describe data and thus eliminate the need to provide pre-defined schemas. The integrated view in TSIMMIS is expressed as a set of views on the data sources (GAV). A more flexible approach, GLAV, is proposed in [FLM99]. The differences between LAV, GAV and GLAV are further discussed in [Len02]. One deficiency of classical data integration solutions is the need for high upfront effort to semantically integrate all source schemas and provide a global mediated schema. Only after this startup cost, queries may be answered using reformulation algorithms [Hal01]. Quality-driven information integration [NLF99] enriches query processing in a mediation system by associating quality criteria to data sources and ranking query plans based on their quality factors. However, not only schema integration must still be performed, but also collection of quality information for each of the data sources. Peer-to-peer data management [TH04, HHL$^+$03, NOTZ03] tries to alleviate this problem by not requiring semantic integration to be performed against all peer schemas. Nevertheless, the data on a new peer is only available after schema integration is performed with respect to at least one of the other peers in the network. In contrast to all of these approaches, dataspace management systems demand basic querying on all data from the start, with the possibility to add

semantic information in a "pay-as-you-go" fashion.

In Figure 1, the corner on the upper left is occupied by **Data Warehouses**: these systems are optimized for read-only access (read-mostly systems). Furthermore, data warehouses require very high semantic integration efforts (integration of multiple schemas). Some PIM systems, such as SEMEX [DH05a] and Haystack [KBH+05], extend data warehouse and information integration technology. They extract information from desktop data sources into a repository and represent that information in a domain model (ontology). The domain model is a high-level mediated schema over the data sources. These systems focus on creating a queryable, however non-updatable, view on the user's personal information.

The corner on the lower right of Figure 1 is occupied by **Versioning Systems** (e.g., Subversion [Sub], Perforce [Per]). These systems provide strong update guarantees (ACID) but do not perform any semantic integration.

File systems occupy the region on the middle-right, providing weaker update guarantees than versioning systems (e.g., recovery on metadata for journaling file systems but no recovery on file content). There are two major approaches to implementing file systems:

(1) The first strategy tries to implement file systems using heavyweight DBMS technology. This idea was already proposed and implemented as early as in Exodus [CDRS86] and the Shore [CDF+94] object-oriented storage manager. Shore represented a merger of object-oriented database and file system technologies. It provided a tree-structured, Unix-like namespace in which all persistent objects were reachable from a distinguished root object. The latter approaches influenced many other research projects, however, they never found acceptance in the operating systems community. Recently, Microsoft WinFS [WFS] attempted to continue that tradition by implementing a file system using a relational DBMS. However, that project was dropped in 2006[1]. Another approach striving to implement a file system using a DBMS is [HGS07] by representing file and folder objects as a huge XML document and storing it in a commercial XML database.

(2) The second strategy to implement file systems is to code them from scratch using only a few lightweight DB techniques such as B^+-trees and simplified recovery techniques (journaling). Today, this strategy is followed by almost all popular file systems including HPFS, XFS, ZFS, EXT3, ReiserFS, and NTFS.

Other approches that do not try to reimplement current file systems but rather enrich file system using XML technolgies are Hubble [LHHB05], and the first version of `iMeMex` [DSKB05]. In this initial version of `iMeMex` we used XML and XQuery to join files (e.g., an Excel sheet and a text file) into a a new file (e.g., a Word document). The join result appeared as a virtual file on a network share.

In Figure 1, the upcoming operating system Windows Vista is also displayed as it provides some basic information managing capabilities (dotted box on the upper right corner), covering functionalities currently offered by file systems and DSEs. However, as Figure 1 shows, Windows Vista covers only a small fraction of the design space covered by Personal Dataspace Management Systems such as `iMeMex`.

[1] The downloadable beta as well as all other preliminary information about WinFS were recently removed from its web-site [WFS].

In summary, Figure 1 shows that a huge design space between the different extremes (sitting in the corners and along the margins) is not covered by current information management solutions. However, in order to be able to manage the entire dataspace of a user that space has to be covered. **PDSMSs** fill that space. These systems cover the entire design space of information systems requiring medium to low semantic integration efforts. In particular, PDSMSs occupy the middle-ground between a read-only DSE (drawbacks: no update guarantees, no information integration, neither physical nor logical data independence), a write-optimized DBMS (drawbacks: schema-first, full-control on data required), and a traditional information integration system (drawback: schema-first).

3 iMeMex Core Architecture

In this section, we discuss the core architecture of the iMeMex PDSMS. iMeMex is based on a layered architecture which is described in Section 3.1. Following that, Sections 3.2 and 3.3 discuss important services that are provided by the different layers.

3.1 Logical Layers

The DSSP vision of Franklin et.al. [FHM05] defines a dataspace as a set of participants (or data sources) and relationships among the participants. We term the set of data sources *Data Source Layer*. Although Franklin et al. [FHM05] present services that should be provided by a DSSP, little is said on how a DSSP would provide those services on top of the Data Source Layer. In fact, in the current state-of-the-art for personal information management, applications (e.g., search&browse, email, Office tools, etc.) access the Data Source Layer (e.g., file systems) directly. This comes at the cost of physical data dependence, including system dependence. This situation is depicted on the left of Figure 2.

To remedy that situation, we argue that what is missing is a logical layer between the applications and the Data Source Layer that provides services on the dataspace. We propose to add the iMeMex PDSMS as that intermediate logical layer. It is depicted on the right of Figure 2. iMeMex abstracts from the underlying subsystems, from data formats, and from devices, providing a coherent view to all applications. iMeMex, however, does not have full control of the data as it is the case with DBMSs. Thus, applications may also access the data sources bypassing iMeMex, e.g., email or office applications do not have to be rewritten to interact with iMeMex: they work directly with the data sources. Other applications, however, may be rewritten to directly operate on iMeMex, e.g., explorer and tcsh.

In the following, we discuss the characteristics of each layer of the iMeMex PDSMS as well as of the layers with which it interacts. All of these layers are shown on the right of Figure 2.

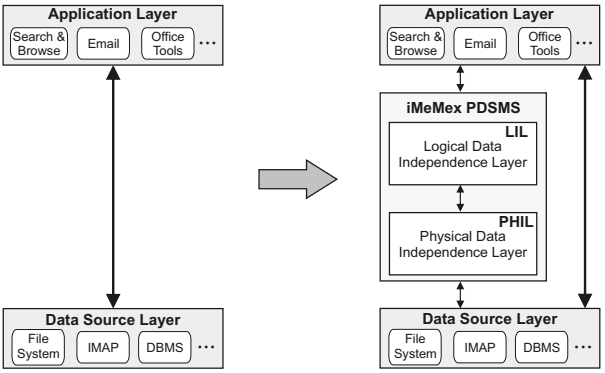

Figure 2: iMeMex remedies the current state-of-the-art in PIM by introducing logical layers that abstract from underlying subsystems, from data formats, and from devices.

Data Source Layer. This layer represents all subsystems managed by the PDSMS. A subsystem that participates on the dataspace may offer either an API that enables full access to the data on that subsystem, access through querying only, or a hybrid of these two options. Thus, the PDSMS must be aware of data vs. query shipping trade-offs [Kos00] to enable efficient query processing.

Physical Data Independence Layer (PHIL). This layer is responsible for resolving the data model, access protocol, and format dependence existing on the data sources participating in the dataspace. PHIL offers unified services such as *data model integration* and *indexing and replication*. We provide more details on these services in Section 3.2.

Logical Data Independence Layer (LIL). This layer provides view definition and query processing capabilities on top of PHIL. LIL offers services such as *result caching*, *view materialization* and *dataspace navigation* for views defined on top of the data unified by PHIL. We discuss important aspects of these services in Section 3.3.

Application Layer. This layer represents the applications built on top of the iMeMex PDSMS. As a PDSMS does not obtain full control of the data, applications may choose to either benefit from the services offered by the PDSMS or access the underlying data sources and use specialized APIs. To enable legacy applications to directly interface with the PDSMS, a PDSMS may offer a mechanism for integrating seamlessly into the host operating system, as demonstrated in [DSKB05].

3.2 PHIL Services

The primary goal of PHIL is to provide physical data independence. Thus, PHIL unifies data reachable in distinct physical storage devices, access protocols and data formats. We present the main services offered by PHIL below.

Data Model Integration. Data model integration refers to the representation of all data available in the data source specific data models using a common model: the *iMeMex Data Model (iDM)* [DS06]. In a nutshell, iDM models each piece of personal information by fine-grained logical entities. These entities may describe files, structural elements inside files, tuples, data streams, XML, or any other piece of information available on

the data sources. These logical entities are linked together in a graph and logically represent the entire personal dataspace of a given user. The details of our data model are beyond the scope of this paper (please see [DS06]). One aspect of this work is that we favor a clear separation of the *logical data model* describing the structural properties of data (flat, relational, tree-structured, graph-structured) and the different possible *physical data representations* (binary, Tables, Object graphs, XML). In the remainder of this paper we use the terms **resource view** and **resource view graph** to refer to a logical piece of information (e.g., an email message, a section in a document, a document, an RSS news entry, etc.), and a graph of logical pieces of information, respectively. Please note, that the `iMeMex` approach is in sharp contrast to *semantic integration*, in which expensive up-front investments have to be made in schema mapping, in order to make the system useful. We follow a pay-as-you-go philosophy [FHM05], offering basic services on the whole dataspace regardless of how semantically integrated the data is. We are currently developing a powerful framework for pay-as-you-go integration on top of our data model.

Indexing and Replication. Given a logical data model to represent all of one's personal information, the next research challenge is how to support efficient querying of that representation. One may consider a pure mediation approach, in which all queries are decomposed and pushed down to the data sources. Though this strategy may be acceptable for local data sources, it may incur long delays when remote data sources are considered. In order to offer maximum flexibility, PHIL offers a hybrid approach. Our approach is based on a *tunable mechanism to bridge warehousing and mediation*. For example, we may choose to replicate relationships among resource views that come from remote data sources, but neither index nor replicate their content. In this situation, relationship navigation among resource views can be accelerated by efficient local access to the replica structures, while retrieval of resource view content will incur costly access to (possibly remote) data sources.

3.3 LIL Services

The primary goal of LIL is to provide logical data independence. LIL enables posing complex queries on the resource view graph [DS06] offered by PHIL. We discuss the services provided by LIL in the following paragraph.

Personal Dataspace Search&Query Language. LIL processes expressions written in a new search&query language for schema-agnostic querying of a resource view graph: the *`iMeMex` Query Language (iQL)*. In our current implementation, the syntax of iQL is a mix between typical search engine keyword expressions and XPath navigational restrictions. The semantics of our language are, however, different from those of XPath and XQuery. Our language's goal is to enable querying of a resource view graph that has not necessarily been submitted to semantic integration. Therefore, as in content and structure search languages (e.g. NEXI [TS04]), our goal is to account for impreciseness in query semantics. For example, by default, when an attribute name is specified (e.g. size>10K), we should not require exact matches on the (implicit or explicit) schema for that attribute, but rather return fuzzy, ranked results that best match the specified conditions (e.g. size,

fileSize, docSize). This allows us to define malleable schemas as in [DH05b]. A PDSMS, however, is not restricted to search. Other important features of iQL are the definition of extensible algebraic operations such as joins and grouping (see [DS06]).

Result Caching. The caching of query results is used to speed up the computation of views. iMeMex's approach to query processing is based on lazy evaluation: whenever matching results are present in the Data Source Layer, PHIL, and/or LIL, then these results should be retrieved from the highest of those layers. However, in this scenario, the freshness of the data may be lower at higher levels in the query processing stack. As a consequence, query processing must take QoS concerns (e.g., freshness) into consideration. Our goal is to deliver stale results quickly and then update the result list as fresh data is delivered from the data sources.

Dataspace Navigation. Users of information systems typically do not start with a precise query specification, but rather develop one in the course of querying and observing results. We call the process of refining query conditions based on a previous definition of the query *dataspace navigation*. It is a common pattern in the exploration of personal information but also data warehousing [DKK05]. In general, if any given set of views were previously computed and had their results cached at LIL, the research challenge is to detect whether a new query may be answered using those views [Hal01]. In difference to [Hal01], these techniques have to work on arbitray content represented as a resource view graph [DS06].

4 iMeMex Features

4.1 Current Features

In this section we present current features of our system as of December 2006 (v 0.42.0).

1. The iMeMex server is implemented in Java 5 and is platform independent. It currently consists of approximately 50,000 Lines of Code and 530 classes.
2. iMeMex is based on a service-oriented architecture as defined by the OSGi framework (similar to Eclipse). This means that services, e.g., data source plugins or content converters, can be exchanged at runtime. Our server may be run with two different OSGi implementations: Equinox [Equ] or Oscar [Osc].
3. All data is represented using the *iMeMex Data Model* [DS06].
4. Our query parser supports an initial version of iQL. Our language iQL supports a mix of keyword and structural search expressions.
5. We provide a rule based query processor that is able to operate in three different querying modes: warehousing (only local indexes and replicas are queried), mediation (local indexes are ignored, queries are shipped to the data sources), and hybrid (combination of the former methods).
6. We provide several different indexing strategies implemented on top of a relational Java database (Apache Derby [Der]) and a full-text search engine (Apache Lucene [Luc]). The relational portions of resource views are vertically decomposed [CK85, ASX01]

to provide better response times. Our primary target is to develop indexes that operate on external memory. However, some of our index structures are main memory resident. Which indexes to use is fully configurable.

7. Scalability: our current version is able to handle up to 25 GB of indexed data (net size, excluding image or music content) on a single iMeMex instance. The biggest file indexed was 7 GB.
8. We have implemented wrapper plugins for the following data sources:
    1. File systems (platform independent: works for Windows, Linux and MAC OS X)
    2. Network shares (SMB)
    3. Email servers (IMAP)
    4. Databases (JDBC, tested with MySQL and Oracle)
    5. Web documents (RSS/ATOM, i.e., any XML data that is accessible by a URI)
9. We provide content converters for LaTeX, Bibtex, XML (SAX-based), and PDF.
10. `iMeMex` provides two important interfaces:
    1. A text console that allows to perform all administration tasks and also allows to query the server.
    2. An HTTP server supporting three different data delivery modes: HTML, XML, and binary. We are also developing an `iMeMex` client that accesses the `iMeMex` server through the HTTP interface. The current version of that client was presented at CIDR 2007 [BDG+07].
11. The `iMeMex` server is open source (Apache 2.0 License) since December 2006 and a first version of our server can be downloaded from `http://www.imemex.org` or `http://imemex.sourceforge.net`.

4.2 Upcoming Features

We are planning to provide the following features with upcoming releases of our software:

1. Pay-as-you-go information integration based on a new declarative framework
2. OS integration for file events (Mac and Windows, using native libraries and C++)
3. Materialized views
4. Cost-based query optimization
5. Integration of updates from data sources
6. Data replication and sharing framework
7. Support for larger datasets > 25 GB, scaling beyond 1 TB using distributed instances.

Please see our web-site for an updated list of supported and upcoming features.

5 Use Cases

In the following, we show two use-cases of our system and how they are supported by the different layers: QoS driven query processing (Section 5.2), and distributed dataspaces (Section 5.3).

5.1 Query Processing Architecture

The design approach we follow in iMeMex is to implement stacked query processors (see Figure 3). The organization of data managing functionality in a processing stack is related to the vision of RISC-style data management components [CW00]. In contrast to [CW00], which focusses on fine-grained data managers for building DBMSs, we advocate a coarse-grained approach that only requires three different query processors. Still, we clearly favor the use of multiple query processors than to use only a single query processors. One reason is that using three query processors provides a clear separation of concerns which in turn facilitates query processing.

5.2 Use Case: QoS Driven Query Answering

Queries posed to iMeMex may be processed by different strategies according to the following QoS parameters:

Exact vs. Relaxed Matches: queries may request the system to produce either *exact* or *relaxed matches*. When relaxed matches are requested, then queries may be decomposed into many relaxed sub-queries (as, for example, in [AY+04]). The results from these sub-queries will be merged and ranked by the query processors according to the relevance to the query.

Freshness vs. Response Time: queries may request *fresh/high cost* results, *low cost/stale* results or *best-effort* results (i.e. low cost or stale results first, followed by fresh or high cost results). For instance, a stale version of a resource view may be quickly returned from a local cache, an up-to-date version of that same resource view may be returned from a remote data source only after some time.

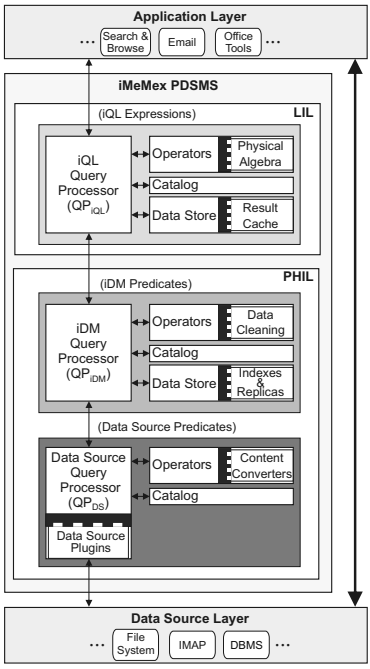

Figure 3: iMeMex is comprised of stacked query processors.

Depending on the QoS parameters for a given query, each query processor assembles a plan that accesses data on the same layer and/or splits and ships a query to the next query processor in the stack. To illustrate that concept, consider the following query which requests all papers in the dataspace au-

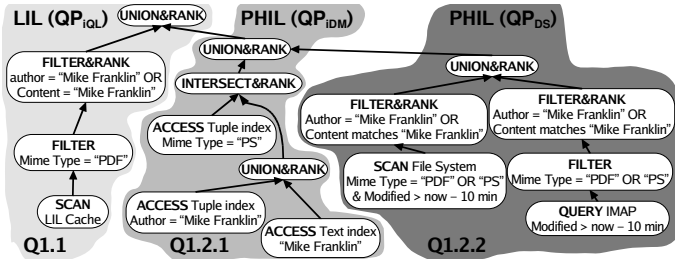

Figure 4: The query plan for query Q1 requesting "all PDF and PS documents authored by Mike Franklin". The figure shows with different colors the plan distribution in different layers (LIL and PHIL) and different query processors that execute different parts of the plan (**QP$_{iQL}$**, **QP$_{iDM}$**, **QP$_{DS}$**).

thored by Mike Franklin:

Q1: `mimetype=(PDF or PS) and author~"Mike Franklin"`

In this example, our personal dataspace includes files in the local file system and emails on the IMAP server. Assume that query

Q2: `mimetype=(PDF or DOC)`

was submitted to the system 10 minutes before Q1 and, consequently, its result is cached at LIL. For simplicity, consider that PHIL was not updated since query Q2. Figure 4 shows the query plan produced by stacked query processors with *relaxed matches* and *best-effort* QoS specifications. The plan on Figure 4 contains three sub-plans, each of them assembled by a different query processor in the stack.

QP$_{iQL}$ starts the planning process and detects that a subset of Q1's results is cached at LIL. This is a consequence of the previous execution of Q2. Since QP$_{iQL}$ has the result for PDF mime types cached, QP$_{iQL}$ rewrites Q1 into two sub-queries:

Q1.1: `mimetype=PDF and author~"Mike Franklin"`

Q1.2:
```
(mimetype=PS and author~"Mike Franklin")
  or (mimetype=PDF and author~"Mike Franklin"
      and modified > now - 10min)
```

The sub-query Q1.1 is planned by QP$_{iQL}$ and its plan is depicted in Figure 4 in the left region. QP$_{iQL}$ plans to scan the cache, filter all resource views that have mime type PDF, and then to filter&rank all resource views having either associated tuples with name `author` and value "Mike Franklin" or content matching "Mike Franklin". In addition, QP$_{iQL}$ plans to union&rank the results from Q1.1's sub-plan with the results of Q1.2. Note that Q1.2, which requests data modified within the last 10 min, is not planned by QP$_{iQL}$, but rather shipped to the next query processor in the stack, namely, QP$_{iDM}$.

QP$_{iDM}$ verifies that, since the indexes and replicas at PHIL were not updated for the last 10 minutes, the up-to-date data has to be requested from the next query processor in the stack. We assume that QP$_{iDM}$ was configured to represent, in its indexes and replicas, all

data in the data sources and therefore may partially answer Q1.2. Thus, QP_{iDM} splits query Q1.2 into two sub-queries:

Q1.2.1: `mimetype=PS and author~"Mike Franklin"`

Q1.2.2: `mimetype=(PDF or PS) and author~"Mike Franklin"`
`and modified > now - 10min`

The sub-query Q1.2.1 is planned by QP_{iDM} and accesses the indexes and replicas in PHIL. The plan for sub-query Q1.2.1 is shown in the middle of Figure 4. It accesses the text indexes for matches of "Mike Franklin" and accesses the tuple index for matches of the tuple `author="Mike Franklin"`. Then it unions and ranks the results from the two index access operators. Afterwards, the plan intersects and ranks the results from the union with the results obtained by accessing the tuple index for matches of the condition `mimetype=PS`. The final operation in the plan is to apply union and ranking to the results obtained from the intersection with the results of sub-query Q1.2.2. Note that sub-query Q1.2.2 is shipped to the next query processor, QP_{DS}.

QP_{DS} is the bottom query processor in `iMeMex`'s query processor stack. The plan it assembles for Q1.2.2 is depicted on the right of Figure 4. The leaves of the query execution plan constructed by QP_{DS} are queries to the data sources. These queries are expressed in the languages specific to the data sources' query interfaces. The leaf on one branch of the plan scans the file system for all PDF or PS files modified in the last 10 minutes. The leaf of the other branch of that plan submits a query to the IMAP server requesting all mails that were received in the last 10 minutes. The results from the IMAP server then pass through a filter that keeps all attachments that are PDF or PS. As Figure 4 shows, the results on both branches are filtered and ranked according to the condition `author="Mike Franklin"` and content matching `"Mike Franklin"`, before feeding them to a union operator that unites the two branches[2], and also ranks the results, before passing them to QP_{iDM}. Each ranking operator ranks according to specific criteria.

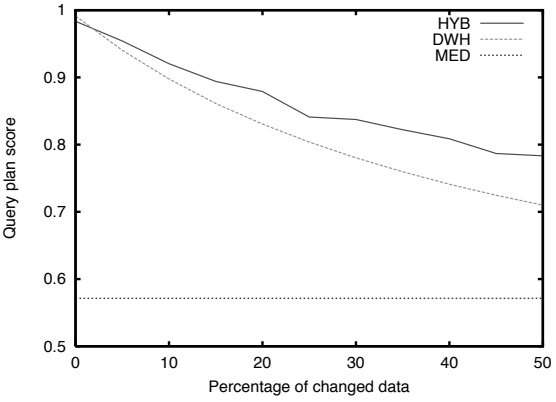

Figure 5: Query execution in a dataspace trading freshness for query response time. The figure shows the score of different query plans as a function of the data updated on the data sources. Three different planning modes are displayed: DWH: data warehousing mode; MED: query mediation mode; HYB: combination of DWH and MED.

[2]An alternative plan would be to retrieve all data from the IMAP server and the local file system that was modified within the last 10 minutes and use that data to simultaneously update the indexes and replicas of the PHIL.

Execution Time vs. Result quality Traditional query processors perform cost-based query optimization by estimating the *cost* of a certain plan. Cost typically only includes an estimate on the execution time. In iMeMex we plan to take a different approach: we want to assess plans using a trade-off function that combines the estimated execution cost with the estimated query result quality. A trade-off function scores a plan and looks as follows:

$$\text{Score}(Plan) = \frac{1}{a+b}\left[a \times \left(1 - \frac{\text{estimated execution time}}{\text{max execution time}}\right) + b \times \frac{\text{\# results}}{\text{max number results}}\right].$$

In this function a and b are weights that determine how much an accurate result has to be favored over a quick result. Figure 5 shows an experiment using a trade-off cost function for three different query modes ($a = 3$, $b = 4$). The horizontal axis depicts the percentage of updates occuring on the data sources. The vertical axis depicts the score as computed by the trade-off function. We executed the query in three different planning modes: (1) DWH: data warehousing mode (only local stale indexes are used by iMeMex, no updates on data sources are considered); (2) MED (local indexes are ignored by iMeMex, the query is fully shipped to the data sources); and (3) HYB (local indexes are exploited by iMeMex but merged with updates from data sources). Figure 5 shows that for very low update rates a DWH-like plan gives the best results w.r.t. the scoring function, for higher update rates (up to 50% in the figure) hybrid query plans attain the highest scores. Pure mediation plans achieve only a constant low score for all update rates as they require costly query processing on the source systems.

5.3 Use Case: Distributed Dataspaces

This section briefly illustrates the distributed query processing capabilities of iMeMex. All data sources that are part of a dataspace managed by iMeMex may be distributed among several machines. For instance, one iMeMex instance may manage N data sources. However, an iMeMex instance, say B, may also play the role of a data source. For instance, an iMeMex instance A may observe iMeMex instance B as a data source. Then, when a query is received by A, that query has to be shipped to B which in turn ships it to

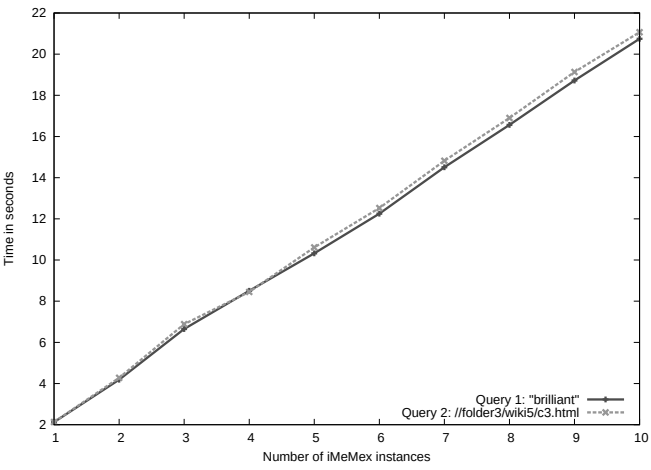

Figure 6: Scaleup experiment for a distributed dataspace scenario: query execution time as a function of the number of iMeMex instances.

its data sources. In distributed networks of iMeMex instances, query planning has to be aware of possible overlap of query results, e.g., two iMeMex data sources may deliver the same results. Further, infinite query shipping loops have to be avoided, i.e., two iMeMex instances may register each other as a data source.

We document the current capabilities of our system by showing a scalability experiment. Figure 6 shows an experiment evaluating the scaleup of a distributed dataspace scenario. In this experiment we installed ten iMeMex instances $I_1, \ldots, I_{10}$ on ten different machines. Those ten iMeMex instances were registered as data sources to another iMeMex instance I_{11}. Instances $I_1, \ldots, I_{10}$ contained identical data sets. The figure displays the query execution time as a function of the number of iMeMex instances appearing in the dataspace. The execution time was measured at I_{11}. Due to space constraints we display two representative queries: query Q1 is a keyword query while Q2 is a path query. The results show that iMeMex scales linearly with the number of iMeMex instances.

6 Conclusions

This paper has advocated the design of a single system to master the personal information jungle [DSKB05]. We have proposed *Personal Dataspace Management Systems (PDSMS)* as a unified solution to manage the entire personal dataspace. This paper reported the current state of the iMeMex project. Our system iMeMex introduces a logical layer on top of the data sources that provides full physical and logical personal information independence. Moreover, iMeMex provides seamless transition between warehousing and information integration. Our PDSMS is based on stacked query processors. We have discussed some of the advantages and challenges of implementing this type of architecture using two use cases. iMeMex is open source (Apache 2.0 License) since December 2006 and an initial version (v0.42.0) of our server can be downloaded from `http://www.imemex.org` or `http://imemex.sourceforge.net`. As part of future work we will improve the existing features and provide new features as presented in Section 4.

Acknowledgements First of all, we would like to thank the anonymous referees for their helpful comments. Furthermore, we would like to thank all M.Sc. students who did their semester project in the iMeMex project (currently about fourteen). All of them contributed to the project and shaped it to its current state.

References

[AAB+03] S. Abiteboul, R. Agrawal, P.A. Bernstein, M.J. Carey, and others. The Lowell Database Research Self Assessment. *The Computing Research Repository (CoRR)*, cs.DB/0310006, 2003.

[ACHK94] Y. Arens, C.Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1994.

[AMS] `http://www.apple.com/macosx/features/spotlight`. Apple Spotlight.

[ASX01] R. Agrawal, A. Somani, and Y. Xu. Storage and Querying of E-Commerce Data. In *VLDB*, 2001.

[AY+04] S. Amer-Yahia et al. FleXPath: Flexible Structure and Full-Text Querying for XML. In *ACM SIGMOD*, 2004.

[BDG+07] L. Blunschi, J.-P. Dittrich, O. Girard, S. Karakashian, and M. Salles. A Dataspae Odyssey: The iMeMex Personal Dataspace Management System (Demo Paper). In *CIDR*, 2007.

[Bea] `http://beaglewiki.org`. Beagle.

[Bel05] G. Bell. Keynote: MyLifeBits: a Memex-Inspired Personal Store; Another TP Database. In *ACM SIGMOD*, 2005.

[Bus45] V. Bush. As We May Think. *Atlantic Monthly*, 1945.

[CDF+94] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring Up Persistent Applications. In *ACM SIGMOD*, 1994.

[CDRS86] M. J. Carey, D. J. DeWitt, J. E. Richardson, and E. J. Shekita. Object and File Management in the EXODUS Extensible Database System. In *VLDB*, 1986.

[CK85] G.P. Copeland and Setrag Khoshafian. A Decomposition Storage Model. In *ACM SIGMOD*, 1985.

[CRDS06] E. Cutrell, D.C. Robbins, S.T. Dumais, and R. Sarin. Fast, flexible filtering with Phlat — Personal search and organization made easy. In *CHI*, 2006.

[CW00] S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *VLDB*, 2000.

[Der] `http://db.apache.org/derby`. Apache Derby.

[DH05a] X. Dong and A. Halevy. A Platform for Personal Information Management and Integration. In *CIDR*, 2005.

[DH05b] X. Dong and A. Halevy. Malleable Schemas: A Preliminary Report. In *WebDB*, 2005.

[DKK05] J.-P. Dittrich, D. Kossmann, and A. Kreutz. Bridging the Gap between OLAP and SQL. In *VLDB*, 2005.

[DS06] J.-P. Dittrich and M. Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *VLDB*, 2006.

[DSKB05] J.-P. Dittrich, M. Salles, D. Kossmann, and L. Blunschi. iMeMex: Escapes from the Personal Information Jungle (Demo Paper). In *VLDB*, 2005.

[Equ] `http://www.eclipse.org/equinox/` Equinox: Eclipse OSGI implementation.

[FG96] E. Freeman and D. Gelernter. Lifestreams: A Storage Model for Personal Data. *SIGMOD Record*, 25(1):80–86, 1996.

[FHM05] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, 2005.

[FLM99] M. Friedman, A. Levy, and T. Millstein. Navigational Plans For Data Integration. In *AAAI - Proceedings of the National Conference on Artificial intelligence*, 1999.

[GDS] `http://desktop.google.com`. Google Desktop.

[Hal01] A. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HED+03] A. Halevy, O. Etzioni, A. Doan, Z. Ives, J. Madhavan, L. McDowell, and I. Tatarinov. Crossing the Structure Chasm. In *CIDR*, 2003.

[HFM06] A. Halevy, M. Franklin, and D. Maier. Principles of Dataspace Systems. In *ACM PODS*, 2006.

[HGS07] A. Holupirek, C. Gruen, and M. H. Scholl. Melting Pot XML, Bringing File Systems and Databases One Step Closer. In *BTW*, 2007.

[HHL+03] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.

[HRO06] A. Halevy, A. Rajaraman, and J. Ordille. Data Integration: The Teenage Years. In *VLDB*, 2006. Ten-year best paper award.

[JB05] W. Jones and H. Bruce. A Report on the NSF-Sponsored Workshop on Personal Information Management, Seattle, WA, 2005. http://pim.ischool.washington.edu/final%20PIM%20 report.pdf.

[KBH+05] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data. In *CIDR*, 2005.

[Kos00] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[KWF+03] M. Kersten, G. Weikum, M. Franklin, D. Keim, A. Buchmann, and S. Chaudhuri. Panel: A Database Striptease or How to Manage Your Personal Databases. In *VLDB*, 2003.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM PODS*, 2002.

[LHHB05] N. Li, J. Hui, H.-I Hsiao, and K. S. Beyer. Hubble: An Advanced Dynamic Folder Technology for XML. In *VLDB*, 2005.

[LRO96] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, 1996.

[Luc] `http://lucene.apache.org/java/docs`. Apache Lucene.

[Mai06] D. Maier. Charting a Dataspace: Lessons from Lewis and Clark. In *EDBT*, 2006. Keynote.

[Mit05] T. Mitchell. Keynote: Computer Workstations as Intelligent Agents. In *ACM SIGMOD*, 2005.

[NLF99] F. Naumann, U. Leser, and J.C. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *VLDB*, 1999.

[NOTZ03] W.S. Ng, B.C. Ooi, K.-L. Tan, and A.Y. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *IEEE ICDE*, 2003.

[Osc] `http://oscar.objectweb.org/` Oscar: OSGi implementation.

[Per] `http://www.perforce.com/`. Perforce.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *IEEE ICDE*, 1995.

[PIM] SIGIR PIM 2006. `http://pim.ischool.washington.edu/pim06home.htm`.

[SLS+93] K.A. Shoens, A. Luniewski, P.M. Schwarz, J.W. Stamos, and J. Thomas II. The Rufus System: Information Organization for Semi-Structured Data. In *VLDB*, 1993.

[Sub] `http://subversion.tigris.org/`. Subversion.

[TH04] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *ACM SIGMOD*, 2004.

[TS04] A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *INEX Workshop*, 2004.

[WFS] `http://msdn.microsoft.com/data/WinFS`. WinFS.