iDM: A Unified and Versatile Data Model for Personal Dataspace Management*

Jens-Peter Dittrich Marcos Antonio Vaz Salles
ETH Zurich
8092 Zurich, Switzerland
dbis.ethz.ch | iMeMex.org

ABSTRACT

Personal Information Management Systems require a powerful and versatile data model that is able to represent a highly heterogeneous mix of data such as relational data, XML, file content, folder hierarchies, emails and email attachments, data streams, RSS feeds and dynamically computed documents, e.g. ActiveXML [3]. Interestingly, until now no approach was proposed that is able to represent all of the above data in a single, powerful yet simple data model. This paper fills this gap. We present the iMeMex Data Model (iDM) for personal information management. iDM is able to represent unstructured, semi-structured and structured data inside a single model. Moreover, iDM is powerful enough to represent graph-structured data, intensional data as well as infinite data streams. Further, our model enables to represent the structural information available inside files. As a consequence, the artifical boundary between inside and outside a file is removed to enable a new class of queries. As iDM allows the representation of the whole personal dataspace [20] of a user in a single model, it is the foundation of the iMeMex Personal Dataspace Management System (PDSMS) [16, 14, 47]. This paper also presents results of an evaluation of an initial iDM implementation in iMeMex that show that iDM can be efficiently supported in a real PDSMS.

1. INTRODUCTION

1.1 Motivation

Personal information consists of a highly heterogeneous data mix of emails, XML, LaTeX and word documents, pictures, music, address book entries, and so on. Personal information is typically stored in files scattered among multiple file systems (local or network), multiple machines (local desktop, network share, mail server), and most of all different file formats (XML, LaTeX, Office, email formats, etc.).

This work is motivated by the fact that much of the data stored in those files represents information such as hierarchies and graphs which are not exploited by current PIM tools to narrow search and query results. Moreover, there is an artificial gap between the structural information *inside* files and the *outside* structural information established by the files&folder hierarchies employed by the user to classify her files.

This paper proposes an elegant solution to close these gaps. Key to our approach is to map all available data to single graph data model — no matter whether it belongs *inside* or *outside* a file. For instance, our model may be applied to represent at the same time data pertaining to the file system (folder hierarchies) and data pertaining to the contents inside a file (structural graph). Consequently the artifical boundary between inside and outside a file is removed. At the same time, we also remove the boundary between different subsystems such as file systems and IMAP email servers as we map that information to the same graph model. This enables new kinds of queries that are not supported with state-of-the-art PIM tools.

1.2 The Problem

EXAMPLE 1 [INSIDE VERSUS OUTSIDE FILES]
Personal Information includes semi-structured data (XML, Word or other Office documents[1]) as well as graph-structured data (LaTeX). These documents are stored as files on the file system. Consider the following query:

> Query 1: "Show me all LaTeX 'Introduction' sections pertaining to project PIM that contain the phrase 'Mike Franklin'."

The query references information that is partly kept *outside* files on the file system, i.e. all project folders related to the PIM project. Another part of the query references information kept *inside* certain LaTeX files, i.e. all introduction sections containing the phrase 'Mike Franklin'. With current technology, this query cannot be issued in one single request by the user as it has to bridge that *inside-outside file boundary*. The user may only search the file system using simple system tools like grep, find, or a keyword search engine. However, these tools may return a large number of results which would have to be examined manually to determine the final result. Even when a matching file is encountered, then, for structured file formats like Microsoft PowerPoint, the user typically has to conduct a second search inside the file to find the desired information [13]. Moreover, state-of-the-art operating systems do not support at all exploitation of structured information inside the user's documents. The structured information inside the files is in a data cage and cannot be used to refine the query.

Desiderata: What is missing is a technology that enables users to execute queries that bridge the divide between the graph-structured information inside their files and the outside file system. □

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

[1] Open Office has stored documents in XML since version 1.0. MS Office 12 appearing end of 2006 will also enable storage of files using zipped XML.

VLDB '06, September 12-15, 2006, Seoul, Korea.

EXAMPLE 2 [FILES VERSUS EMAIL ATTACHMENTS]
Email has become one of the most important applications of personal information management. Consider a simple project management scenario. When managing multiple projects, e.g. industry projects, research projects, PhD students, lectures, and so on, you may decide to store documents of big projects in a separate folder on your local hard disk. For smaller projects you may decide to keep that information as attachments to email messages you exchanged with members of the project team. Now let's consider the following query:

> Query 2: "Show me all documents pertaining to project 'OLAP' that have a figure containing the phrase 'Indexing Time' in its label."

With state-of-the-art technology this type of query cannot be computed as it requires to consider *structural information* available on the different folder hierachies *outside*, i.e., both the folder hierarchy on the local hard disk as well as the folder hierarchy on the IMAP server. Again, the structural constraint "all Figures containing" has to be evaluated considering *structural information* present *inside* certain files, i.e. locally available files and email attachments.
Desiderata: What is missing is a technology that abstracts from the different *outside* subsystems like file systems, and email servers to enable a unified approach to querying graph-structured personal information. □

1.3 Our Contributions

This paper presents an elegant solution to the above problems. The core idea is to introduce a unified, versatile and yet simple data model. All personal information like semi-structured documents (LaTeX, Word or other Office documents), relational data, file content, folder hierarchies, email, RSS feeds and even data streams can be instantiated in that model. Since we are able to represent all data inside a single model, we are then in the position to use a single powerful query language to query all data within a single query. This paper presents our data model. The full specification of our query language will be presented as part of future work.

In summary, this paper makes the following contributions:

1. We present the *iMeMex Data Model (iDM)* for personal information management. iDM allows for the unified representation of all personal information like XML, relational data, file content, folder hierarchies, email, data streams and RSS feeds inside a single data model.
2. We present how to instantiate existing data like XML, relations, files&folders and data streams inside our model. Further, we show how to instantiate iDM subgraphs based on the structural content of files such as tree structures (XML) and graph structures (LaTeX).
3. We show that all parts of the *iMeMex Data Model (iDM)* can be computed lazily. Because of this, iDM is able to support so-called *intensional data*, i.e. data that is obtained by executing a query or calling a remote service. For this reason, iDM can model approaches such as Active XML [3] as a use-case. In contrast, to the latter, however, iDM is not restricted to the XML domain.
4. iDM is the foundation for the *iMeMex Personal Dataspace Management System (PDSMS)* [16, 14, 47]. A PDSMS provides integrated services on the whole dataspace [20] of a user, that is the total of all personal information pertaining to a given individual. We present the core architecture of iMeMex and show how iDM was implemented in that system.
5. Using an initial iDM implementation in the iMeMex Personal Dataspace Management System we present results of experiments. The result of our experiments demonstrate that iDM can be efficiently implemented in a real PDSMS, providing both efficient indexing times and interactive response times.

This paper is structured as follows. Section 2 presents an overview and the formal definition of the *iMeMex Data Model (iDM)* for personal information management. Section 3 then describes how to represent XML, files&folders, as well as data streams using iDM. After that, Section 4 discusses how to compute an iDM graph. Section 5 gives an overview on the implementation of iDM and its query language iQL in the iMeMex Personal Dataspace Management System. Section 6 reviews related work. Section 7 presents experiments with our initial iDM implementation in iMeMex. Finally, Section 8 concludes the paper.

2. iMeMex DATA MODEL

This section is structured into an overview (2.1), the formal definition of our model (2.2), and a series of examples (2.3).

2.1 Overview

The iMeMex Data Model (iDM) uses the following important ideas to represent the heterogeneous data mix found in personal information management:

- In iDM, all personal information available on a user's desktop is exposed through a set of *resource views*. A resource view is a sequence of components that express *structured*, *semi-structured* and *unstructured* pieces of the underlying data. Thus all personal data items, though varied in their representation, are exposed in iDM uniformly. For example, every node in a files&folders hierarchy as well as every element in an XML document would be represented in iDM by one distinct resource view.

- Resource views in iDM are linked to each other in arbitrary directed *graph structures*. The connections from a given resource view to other resource views are given by one of its components.

- In contrast to XML approaches [45, 3], iDM does not impose the need to convert data to a physical XML document representation before query processing may take place. Rather, we favor a clear *separation between logical and physical representation* of data. Data may be dynamically represented in our model during query processing although it remains primarily stored in its original format and location. Note that this does not preclude a system that implements iDM to provide facilities such as indexing or replication of the original data to speed-up query evaluation.

- We introduce *resource view classes* to precisely define the types and representation of resource view components. Any given resource view may or may not comply to a resource view class. We show how to use resource view classes to constrain our model to represent data in files&folders, relations, XML and data streams in Section 3.

- Resource views may be given *extensionally* (e.g., files&folders or tuples in a relational store) or computed *intensionally* (e.g., as a result to a query). Further, resource views may contain components that are *finite* as well as *infinite*. Those aspects of our model are explored in Section 4.

2.2 Resource Views

In this section, we formally define a resource view and explain its constituent parts. After that, we present examples of resource view graphs (see Section 2.3).

DEFINITION 1 (RESOURCE VIEW) *A resource view V_i is a 4-tuple $(\eta_i, \tau_i, \chi_i, \gamma_i)$, where η_i is a name component, τ_i is a tuple component, χ_i is a content component, and γ_i is a group component*[2]*. We define each component of a resource view V_i as follows:*

- η_i NAME COMPONENT*: η_i is a finite string that represents the name of V_i.*
- τ_i TUPLE COMPONENT*: τ_i is a 2-tuple (W, T), where W is a schema and T is one single tuple that conforms to W. The schema $W = \langle a_j \rangle, j = 1, 2, \ldots, k$ is defined as a sequence of attributes where attribute a_j is the name of a role played by some domain*[3] *D_j in W. The tuple $T = \langle v_j \rangle, j = 1, 2, \ldots, k$ is a sequence of atomic values where value v_j is an element of the domain D_j of the attribute a_j.*
- χ_i CONTENT COMPONENT*: χ_i is a string of symbols taken from an alphabet Σ_c. The content χ_i may be finite or infinite. When χ_i is finite, it takes the form of a finite sequence of symbols denoted by $\langle c_1, \ldots, c_l \rangle, c_j \in \Sigma_c, j = 1, \ldots, l$; when χ_i is infinite, the respective sequence is infinite and we denote $\chi_i = \langle c_1, \ldots, c_l \rangle_{l \to \infty}, c_j \in \Sigma_c, j = 1, \ldots, l, l \to \infty$.*
- γ_i GROUP COMPONENT*: γ_i is a 2-tuple (S, Q), where S is a (possibly empty)* ***set*** *of resource views and Q is a (possibly empty) ordered* ***sequence*** *of resource views. Further,*
  - *(i) The set S and the sequence Q may be finite or infinite. When S is finite, we denote $S = \{V_{s_1}, \ldots, V_{s_m}\}$; when it is infinite, then we denote $S = \{V_{s_1}, \ldots, V_{s_m}\}_{m \to \infty}$. Likewise, when Q is finite, we denote $Q = \langle V_{q_1}, \ldots, V_{q_n} \rangle$; when Q is infinite, then we denote $Q = \langle V_{q_1}, \ldots, V_{q_n} \rangle_{n \to \infty}$.*
  - *(ii) $S \cap Q = \emptyset$, i.e., S and Q are disjoint*[4]*.*
  - *(iii) Assume a resource view V_i has a non-empty γ_i component. If there exists a resource view V_k for which $V_k \in S \cup Q$ holds, we say that V_k is* ***directly related*** *to V_i, i.e., $V_i \to V_k$. Any given resource view may be directly related to zero, one or many other resource views.*
  - *(iv) If $V_i \to V_j \to \ldots \to V_k$, we say that resource view V_k is* ***indirectly related*** *to V_i, i.e., $V_i \rightsquigarrow V_k$.*

If any of the components of a resource view is empty, we denote its value, where convenient, by the empty n-tuple () or by the empty sequence $\langle \rangle$. □

The η_i component is a name used to refer to the resource view. It is of service for the construction of path queries, discussed in Section 5.1. The τ_i component has a similar definition as the one given for tuples in a relational data model [9, 19]. One important difference is that the schema W is defined for *each* tuple, instead of for a set of tuples. This decision of course creates a tension for representing sets of resource views with the same structural features. Schematic information is important in this setting and we introduce it in our model by defining *resource view classes* (see Section 3).

The χ_i component represents arbitrary unstructured content. For iDM, it is merely a sequence of atomic symbols from some alphabet, such as characters in a file's content or in an XML text node. The χ_i component may be finite or infinite. Allowing infinite content is useful to naturally represent media streams in our model.

Finally, the γ_i component creates a directed graph structure that connects related views. Note that we impose no restriction on this graph, so that we may represent trees, DAGs and cyclic graphs. If the relative order among the connections established between resource views is of importance, we represent them in the sequence Q of γ_i. Otherwise, they are represented in the set S. Like the χ_i component, both the set and/or the sequence of γ_i may be finite or infinite. The infinite case is useful for representing data streams as infinite sequences of resource views in our model.

2.3 Examples

In Figure 1(a), we present a typical files&folders hierarchy containing information on some research projects of one of the authors. There is one high-level folder that groups all research projects and we show two sub-folders for specific projects. The 'PIM' folder is further expanded to reveal one LaTeX document for one version of this VLDB 2006 paper, one Microsoft Word document for a grant proposal, and one folder link to the top-level 'Projects' folder. The contents of those documents are also partially displayed. Both documents contain sections, subsections and their corresponding text. In addition, the document 'vldb2006.tex' also contains a reference to section 'Preliminaries' in the subsection 'The Problem'.

Unified Representation. As we may notice in Figure 1(a), there is a gap between the graph-structured information *inside* files and the hierarchies present on the *outside* file system. We show the representation of the same information in iDM in Figure 1(b). Nodes denote resource views and edges denote the connections induced by the resource views' group components. Each node is labeled with the resource view's name component.

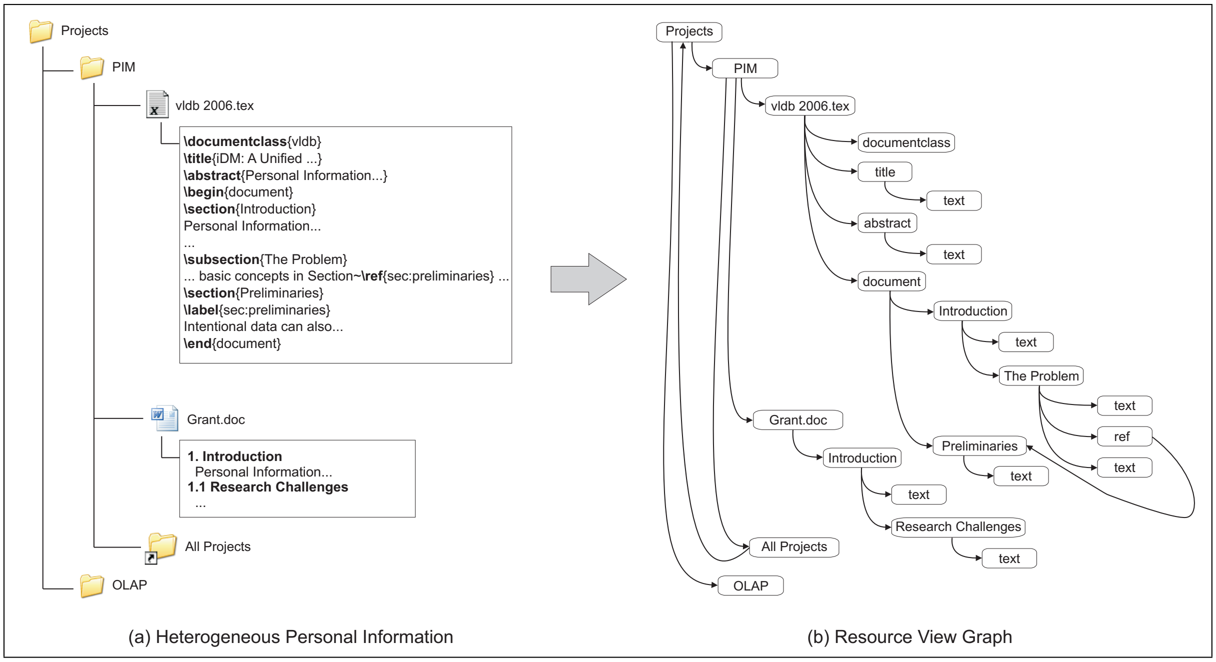
In iDM, what we call files&folders are only resource views. Each file or folder is represented as one resource view in Figure 1(b). Further, the data stored inside files is also uniformly represented as resource views. The document class, title, abstract and document portions of the VLDB 2006 paper are some of these resource views. They are directly related to the 'vldb 2006.tex' file resource view. In addition, any structural information present in the document portion of that file is represented as resource views. Note that the same applies to the 'Grant.doc' resource view. This is possible as an increasing number of office tools, such as Microsoft Office 12, Open Office and LaTeX, offer semi-structured or graph-structured document formats that make it feasible to write content converters that obtain good quality resource view graphs. In addition, structure extraction techniques [18, 12, 35] may be used in conjunction with our approach to further increase the quality of resource view subgraphs extracted from content components.

Intensional Data. One important aspect of our model is that the resource view graph is a *logical* representation of personal information. That logical representation does not need to be fully materialized at once and may be computed lazily. Personal information does not have to be imported to or exported from our data model, but rather represented in it. This is in sharp contrast to XML approaches which are usually tied to having a physical representation of the whole data as an XML document before querying may be carried out (see Section 4).

[2] We use the subscript notation to denote one specific resource view and its components.

[3] A domain is considered to be a set of atomic values. In the remainder, whenever we mention attributes or domains, we refer to the definitions given above. Our definitions of domains and attributes conform to the ones given in [19].

[4] For the sake of simplicity, we extend notation and use the symbols $\cap$ and $\cup$ to denote intersection not only between sets, but also between a set and a sequence. The $\cap$ operation returns a set containing the (distinct) elements of the sequence that also belong to the intersected set. The $\cup$ operation returns a set containing the (distinct) elements that belong either to the set or to the sequence.



```
\documentclass{vldb}
\title{iDM: A Unified ...}
\abstract{Personal Information...}
\begin{document}
\section{Introduction}
Personal Information...
...
\subsection{The Problem}
... basic concepts in Section~\ref{sec:preliminaries} ...
\section{Preliminaries}
\label{sec:preliminaries}
Intentional data can also...
\end{document}
```

```
1. Introduction
  Personal Information...
1.1 Research Challenges
  ...
```

Figure 1: iDM represents heterogeneous personal information as a single resource view graph. The resource view graph represents the whole dataspace of a user

Resource Views and their Components. Let us show the detailed representation of one resource view present in Figure 1. Consider the 'PIM' folder. We represent it as a resource view $V_{\mathsf{PIM}} = (\eta_{\mathsf{PIM}}, \tau_{\mathsf{PIM}}, \chi_{\mathsf{PIM}}, \gamma_{\mathsf{PIM}})$, in which:

$\eta_{\mathsf{PIM}} =$ 'PIM';

$\tau_{\mathsf{PIM}} =$ (W,T), where $W = \langle$creation time: date, size: integer, last modified time: date$\rangle$ and $T = \langle$'19/03/2005 11:54', 4096, '22/09/2005 16:14'$\rangle$, i.e., the τ_{PIM} component represents the filesystem properties associated with the 'PIM' folder;

$\chi_{\mathsf{PIM}} = \langle\rangle$;

$\gamma_{\mathsf{PIM}} =$ (S,Q), where $S = \{V_{\mathsf{vldb\ 2006.tex}}, V_{\mathsf{Grant.doc}}, V_{\mathsf{All\ Projects}}\}$ and $Q = \langle\rangle$.

Note that the children of the 'PIM' folder in the filesystem are represented as resource views directly related to V_{PIM}. These resource views are $V_{\mathsf{vldb\ 2006.tex}}$, $V_{\mathsf{Grant.doc}}$, and $V_{\mathsf{All\ Projects}}$. The resource views $V_{\mathsf{vldb\ 2006.tex}}$ and $V_{\mathsf{Grant.doc}}$ have their η and τ components defined analogously to V_{PIM}, their γ component corresponding to the resource views in their document content and their χ component equal to the binary stream of each file. The $V_{\mathsf{All\ Projects}}$ resource view is also represented analogously to V_{PIM}, except that its γ component corresponds to the folder 'Projects'. Here and in the remainder, we will omit the empty components from the resource view notation as they are clear from the context. Therefore, we denote $V_{\mathsf{PIM}} =$ ('PIM', $\tau_{\mathsf{PIM}}, \gamma_{\mathsf{PIM}}$). We also use V_{PIM} to represent a view named 'PIM' where there is no risk of confusion with other views with the same name.

Graph Structures. Some data models, such as XML and traditional files&folders, organize data in a tree structure. The extension of that structure to represent graphs is usually done through the inclusion of *second-class citizens* in the model, such as links. iDM naturally represents graph structures by the connections induced by the resource views' group components. The relevance of representing and querying arbitrary graph data has been increasingly recognized in the literature [24, 30].

For example, the $V_{\mathsf{Projects}} \rightarrow V_{\mathsf{PIM}} \rightarrow V_{\mathsf{All\ Projects}} \rightarrow V_{\mathsf{Projects}}$ path of directly related resource views in Figure 1(b) forms a cycle in the resource view graph. Further, in Figure 1(b), resource view $V_{\mathsf{Preliminaries}}$ is directly related to both views V_{document} and V_{ref}.

Schemas. Personal information is trapped inside a large array of data cages. Some solutions to this problem propose to cast user's information into a rigid set of developer defined schemas, e.g. Microsoft WinFS [44]. They defend that the relational model is the most appropriate underlying representation for desktop data. Other proposals, e.g. Apple Spotlight [4], abolish schemas and employ search engine technology to represent all data as completely unstructured.

In contrast, iDM offers a schema-agnostic graph representation of personal information. We may employ resource view classes, defined in the following section, to enrich iDM with schematic information and to constrain it to represent a wide array of data models typically used to represent personal data.

3. INSTANTIATING SPECIALIZED DATA MODELS

In the following, we show how to instantiate specialized data models like files&folders, XML, as well as data streams using iDM. Due to space constraints, we omit details on how to instantiate relational data. These instantiations will, however, become clear to the reader based on the discussion for the other data models. Table 1 summarizes the instantiations for the specialized data models.

3.1 Resource View Classes

For convenience, we introduce *resource view classes*, which constrain iDM to represent a particular data model.

DEFINITION 2 (RESOURCE VIEW CLASS) *Given a set of resource views $D = \{V_i\}, i = 1, \ldots, n$, we define a resource view class C as a set of formal restrictions on the η_i, τ_i, χ_i and γ_i components of all views $V_i \in D$. The formal restrictions specified by a resource view class include:*

Resource View Class		Resource View Components Definition				
		$\eta_{\text{i}}^{\text{C}}$	$\tau_{\text{i}}^{\text{C}}$	$\chi_{\text{i}}^{\text{C}}$	$\gamma_{\text{i}}^{\text{C}}$	
Description	**Name**				S	Q
File	file	N_f	(W_{FS}, T_f)	C_f	$\varnothing$	$\langle\rangle$
Folder	folder	N_F	(W_{FS}, T_F)	$\langle\ \rangle$	$\{V_1^{\text{child}}, \ldots, V_{\text{m}}^{\text{child}}\}$ $\text{child} \in \{\text{file}, \text{folder}\}$	$\langle\rangle$
Relational Tuple	tuple	$\langle\ \rangle$	(W_R, t_i)	$\langle\ \rangle$	$\varnothing$	$\langle\rangle$
Relation	relation	N_R	$(\ )$	$\langle\ \rangle$	$\{V_1^{\text{tuple}}, \ldots, V_{\text{m}}^{\text{tuple}}\}$ $V_{\text{i}}^{\text{tuple}} = \langle\tau_{\text{i}}^{\text{tuple}}\rangle, \tau_{\text{i}}^{\text{tuple}} = (W_R, t_i)$, $i = 1, \ldots, m$	$\langle\rangle$
Relational database	reldb	N_{DB}	$(\ )$	$\langle\ \rangle$	$\{V_1^{\text{relation}}, \ldots, V_{\text{m}}^{\text{relation}}\}$	$\langle\rangle$
XML text node	xmltext	$\langle\ \rangle$	$(\ )$	C_t	$\varnothing$	$\langle\rangle$
XML element	xmlelem	N_E	(W_E, T_E)	$\langle\ \rangle$	$\varnothing$	$\langle V_1^{\text{xmlnode}}, \ldots, V_{\text{n}}^{\text{xmlnode}}\rangle$ $\text{xmlnode} \in \{\text{xmltext}, \text{xmlelem}\}$
XML document	xmldoc	$\langle\ \rangle$	$(\ )$	$\langle\ \rangle$	$\varnothing$	$\langle V_{\text{root}}^{\text{xmlelem}}\rangle$
XML File	xmlfile	N_f	(W_{FS}, T_f)	C_f	$\varnothing$	$\langle V_{\text{doc}}^{\text{xmldoc}}\rangle$
Data Stream	datstream	$\langle\ \rangle$	$(\ )$	$\langle\ \rangle$	$\varnothing$	$\langle V_1, \ldots, V_{\text{n}}\rangle_{n\to\infty}$
Tuple stream	tupstream	$\langle\ \rangle$	$(\ )$	$\langle\ \rangle$	$\varnothing$	$\langle V_1^{\text{tuple}}, \ldots, V_{\text{n}}^{\text{tuple}}\rangle_{n\to\infty}$
RSS/ATOM stream	rssatom	$\langle\ \rangle$	$(\ )$	$\langle\ \rangle$	$\varnothing$	$\langle V_1^{\text{xmldoc}}, \ldots, V_{\text{n}}^{\text{xmldoc}}\rangle_{n\to\infty}$ or: same as in xmldoc

Table 1: Important Resource View Classes to represent files&folders, relations, XML, data streams, and RSS

1. EMPTYNESS OF COMPONENTS*: specifies that a certain subset of the components of resource views that obey to the resource view class must be empty or not.*
2. SCHEMA OF τ*: determines the schema W that the τ components of resource views must have.*
3. FINITENESS OF χ OR γ*: enforces content χ or group γ elements S or Q to be either finite, infinite or empty.*
4. CLASSES OF DIRECTLY RELATED RESOURCE VIEWS*: given a resource view V_{i}, determines the set of resource view classes that are acceptable for any resource view V_{j} such that $V_{\text{i}} \to V_{\text{j}}$.*

When a view V_{i} conforms to the resource view class C, we denote it V_{i}^{C}. Accordingly, we denote its components $\eta_{\text{i}}^{\text{C}}$, $\tau_{\text{i}}^{\text{C}}$, $\chi_{\text{i}}^{\text{C}}$, and $\gamma_{\text{i}}^{\text{C}}$.

A given resource view may obey directly to only one class. One may, however, organize resource view classes in generalization (or specialization) hierarchies. When a resource view obeys to a given class C, it automatically obeys to all classes that are generalizations of C. In this sense, our classes have an object-oriented flavor [8].

Resource view classes may be used by application developers to provide pre-defined schema information on sets of resource views. Note that a full specification of schemas, including several different categories of integrity constraints, exceeds the scope of this paper. Note, additionally, that not all resource views need to have a resource view class attached to them. This means that unlike the relational [9] or object-oriented approaches [8], which support only a schema-first data modeling strategy, our model also supports schema-later and even schema-never [20].

3.2 Files&Folders

Traditional filesystems can be seen as trees in which internal nodes represent non-empty folders and leaves represent files or empty folders. Each node in the tree has a fixed set of properties, such as size, creation time, last modified time, etc. The attributes that appear on each node are defined in a filesystem-level schema $W_{FS} = \{$size: int, creation time: date, last modified time: date, ...$\}$. The sequence of values that conform to that schema are expressed per node and, for a node n, we denote this sequence T_n. Each node also has a name, denoted N_n. In addition, if n is a file node, then it has an associated content C_n. In iDM we consider a 'file' just one out of many possible resource views on user data. In order to have iDM represent the files&folders data model we define a *file resource view class*, denoted file, that constrains a view instance $V_{\text{i}}^{\text{file}}$ to represent a file f as follows:

$$V_{\text{i}}^{\text{file}} = (\eta_{\text{i}}^{\text{file}}, \tau_{\text{i}}^{\text{file}}, \chi_{\text{i}}^{\text{file}}), \text{ where:}$$

$$\eta_{\text{i}}^{\text{file}} = N_f, \tau_{\text{i}}^{\text{file}} = (W_{FS}, T_f), \text{ and } \chi_{\text{i}}^{\text{file}} = C_f.$$

Based on this definition we can recursively define the concept of a 'folder'. A *folder resource view class*, denoted folder, constrains a view instance $V_{\text{i}}^{\text{folder}}$ to represent a folder F as follows:

$$V_{\text{i}}^{\text{folder}} = (\eta_{\text{i}}^{\text{folder}}, \tau_{\text{i}}^{\text{folder}}, \gamma_{\text{i}}^{\text{folder}}), \text{ where:}$$

$$\eta_{\text{i}}^{\text{folder}} = N_F, \tau_{\text{i}}^{\text{folder}} = (W_{FS}, T_F), \text{ and}$$

$$\gamma_{\text{i}}^{\text{folder}} = \big(\{V_1^{\text{child}}, \ldots, V_{\text{m}}^{\text{child}}\}, \langle\rangle\big), \text{child} \in \{\text{file}, \text{folder}\}.$$

Further, as discussed in Section 2, iDM may be used to exploit semi-structured content inside files. One special case of this type of content is XML. We may thus specialize the file resource view class to define an *XML file resource view class*, denoted xmlfile, in which the group component is non-empty and defined as:

$$\gamma_{\text{i}}^{\text{xmlfile}} = \big(\varnothing, \langle V_{\text{doc}}^{\text{xmldoc}}\rangle\big).$$

The resource view $V_{\text{doc}}^{\text{xmldoc}}$ pertains to resource view class xmldoc and is the document resource view for the XML document. The xmldoc resource view class is defined in Subsection 3.3. Note that other specializations of the file resource view class may be defined analogously for other document formats, e.g. LaTeX.

3.3 XML

We assume that XML data is represented according to the definitions in the XML Information Set [46]. Due to space constraints, we only discuss how to instantiate the core subset of the XML Information Set (document, element, attribute, character) in iDM.

In order to have iDM represent the XML data model we first define an *XML text resource view class*, denoted xmltext, that constrains a view instance $V_{\mathsf{i}}^{\mathsf{xmltext}}$ to represent a character information item with text content $C_t = \langle c_1, \ldots, c_n \rangle$ as follows:

$$V_{\mathsf{i}}^{\mathsf{xmltext}} = (\chi_{\mathsf{i}}^{\mathsf{xmltext}}), \text{ where: } \chi_{\mathsf{i}}^{\mathsf{xmltext}} = C_t.$$

As a second step, we need to represent element information items in our model. An element information item E has a name N_E, a set of attributes with values T_E and schema W_E, and a sequence of children, each of which is either a text or an element information item. We define an *XML element resource view class*, xmlelem, that constrains a view instance $V_{\mathsf{i}}^{\mathsf{xmlelem}}$ to represent an element information item E as follows:

$$\begin{aligned}
V_{\mathsf{i}}^{\mathsf{xmlelem}} &= (\eta_{\mathsf{i}}^{\mathsf{xmlelem}}, \tau_{\mathsf{i}}^{\mathsf{xmlelem}}, \gamma_{\mathsf{i}}^{\mathsf{xmlelem}}), \text{ where:}\\
&\eta_{\mathsf{i}}^{\mathsf{xmlelem}} = N_E,\, \tau_{\mathsf{i}}^{\mathsf{xmlelem}} = (W_E, T_E), \text{ and}\\
&\gamma_{\mathsf{i}}^{\mathsf{xmlelem}} = (S, Q), S = \varnothing,\\
&\quad Q = \langle V_1^{\mathsf{xmlnode}}, \ldots, V_{\mathsf{n}}^{\mathsf{xmlnode}} \rangle,\\
&\quad \mathsf{xmlnode} \in \{\mathsf{xmltext}, \mathsf{xmlelem}\}.
\end{aligned}$$

Note that the XML attribute nodes are modelled by the $\tau_{\mathsf{i}}^{\mathsf{xmlelem}}$ component of $V_{\mathsf{i}}^{\mathsf{xmlelem}}$. Finally, we can define an *XML document resource view class*, xmldoc, to represent a document information item as follows:

$$\begin{aligned}
V_{\mathsf{i}}^{\mathsf{xmldoc}} &= (\gamma_{\mathsf{i}}^{\mathsf{xmldoc}}), \text{ where:}\\
&\gamma_{\mathsf{i}}^{\mathsf{xmldoc}} = \big(\varnothing, \langle V_{\mathsf{root}}^{\mathsf{xmlelem}} \rangle\big), \text{ and}
\end{aligned}$$

$V_{\mathsf{root}}^{\mathsf{xmlelem}}$ is a view that represents the root element information item of the document. Figure 2 shows an example of an instantiation of an XML fragment in iDM. Each node in the XML document is represented as a resource view. We use an expanded notation in the figure to represent nodes in the resource view graph and explicitly indicate their components. The connections among views are given by the resource views' γ components.

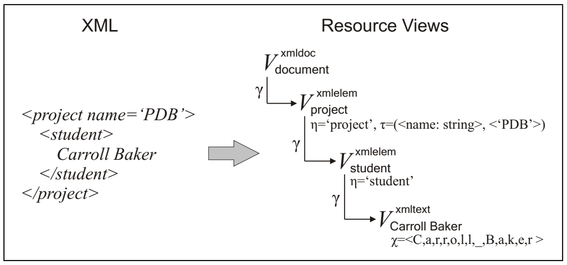

XML
Resource Views
<project name='PDB'>
<student>
Carroll Baker
</student>
</project>
$V_{\mathsf{document}}^{\mathsf{xmldoc}}$
γ
$V_{\mathsf{project}}^{\mathsf{xmlelem}}$
η='project', τ=(<name: string>, <'PDB'>)
γ
$V_{\mathsf{student}}^{\mathsf{xmlelem}}$
η='student'
γ
$V_{\mathsf{Carroll\ Baker}}^{\mathsf{xmltext}}$
χ=<C,a,r,r,o,l,l,_,B,a,k,e,r >

Figure 2: XML fragment represented as a resource view graph

3.4 Data Streams

We assume that a data stream is an infinite data source delivering data items. For the moment we do not require any restrictions on the type of data items. To constrain iDM to represent a generic data stream model we define a **generic data stream** resource view class, datstream, that restricts a resource view $V_{\mathsf{i}}^{\mathsf{datstream}}$ as follows:

$$V_{\mathsf{i}}^{\mathsf{datstream}} = (\gamma_{\mathsf{i}}^{\mathsf{datstream}}), \text{ where: } \gamma_{\mathsf{i}}^{\mathsf{datstream}} = \big(\varnothing, \langle V_1, \ldots, V_{\mathsf{n}} \rangle_{n \to \infty}\big).$$

Figure 3 shows schematically how the instantiation of a data stream occurs in iDM. Each data item delivered by that stream is represented as a resource view. Depending on the data model in which the item is represented, the corresponding resource views may be of different classes. The data stream itself is represented as a resource view whose γ component contains all corresponding data item views.

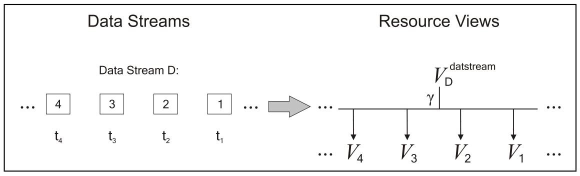


Figure 3: A data stream is represented as a resource view graph

Examples of data streams include streams that deliver tuples. A **tuple stream** resource view class, tupstream, is defined to restrict a view instance $V_{\mathsf{i}}^{\mathsf{tupstream}}$ as follows:

$$\begin{aligned}
V_{\mathsf{i}}^{\mathsf{tupstream}} &= (\gamma_{\mathsf{i}}^{\mathsf{tupstream}}), \text{ where:}\\
&\gamma_{\mathsf{i}}^{\mathsf{tupstream}} = \big(\varnothing, \langle V_1^{\mathsf{tuple}}, \ldots, V_{\mathsf{n}}^{\mathsf{tuple}} \rangle_{n \to \infty}\big).
\end{aligned}$$

Another example of a data stream is an RSS/ATOM stream delivering XML messages. An **RSS stream** resource view class, rssatom, is defined to restrict a resource view $V_{\mathsf{i}}^{\mathsf{rssatom}}$ as follows:

$$\begin{aligned}
V_{\mathsf{i}}^{\mathsf{rssatom}} &= (\gamma_{\mathsf{i}}^{\mathsf{rssatom}}),\\
&\gamma_{\mathsf{i}}^{\mathsf{rssatom}} = \big(\varnothing, \langle V_1^{\mathsf{xmldoc}}, \ldots, V_{\mathsf{n}}^{\mathsf{xmldoc}} \rangle_{n \to \infty}\big).
\end{aligned}$$

Note that RSS/ATOM streams, even though they are frequently called 'streams', are just simple XML documents that are provided by a growing number of web servers. There is no notification mechanism for the clients interested in those documents. So, one may argue that an alternative representation for an RSS/ATOM stream is the same as given for an XML document (see also Table 1).

4. COMPUTING THE iDM GRAPH

This section outlines how to compute resource views and resource view graphs. In the following, we show that all components of a resource view may be computed lazily (Section 4.1). After that, we discuss three paradigms to compute resource view components: *extensional components* (Section 4.2), *intensional components* (Section 4.3), and *infinite components* (Section 4.4).

4.1 Lazy Resource Views

It is important to understand that all components of a resource view may be computed on demand (aka lazily or intensionally). We do *not* require that any of the components of the iDM graph are materialized beforehand. Whether the components of a resource view are materialized or not is hidden by modelling a resource view as an interface consisting of four get-methods, one for each component:

```
Interface ResourceView {
        getNameComponent(): return η
        getTupleComponent(): return τ
        getContentComponent(): return χ
        getGroupComponent() : return γ
}
```

As a consequence, each resource view hides *how*, *when* and even *where* the different components are computed.

For instance, in Figure 1 the subgraph representing the contents of LATEX file "vldb 2006.tex" may be transformed to an iDM graph if a user requests that information, i.e. when she calls getGroupComponent() on resource view $V_{\mathsf{vldb\ 2006.tex}}^{\mathsf{file}}$. In the following sections, we discuss how to obtain the data returned by the four different get*Component-methods.

4.2 Extensional Components

One important class of resource view components are *extensional components*. Extensional components return base facts. No additional query processing is required to retrieve those facts.

A prominent example of base facts is data that is stored on a hard disk drive or in main memory. For instance, the byte content of a Word file is stored on disk. That byte content is considered base facts. A second example is a DB table that is persisted in one of the segments of a DBMS.

4.3 Intensional Components

In contrast to extensional components, *intensional components* require query processing to compute the component. Examples of intensional components include computing the result to a query based on locally available data or based on calling remote hosts.

For instance, a view defined on top of a set of DB tables is considered intensional data. This holds even if that view was explicitly created as a materialized view [25]. In that case that data was simply precomputed (aka materialized). However, on the logical iDM level that data is still considered intensional data as it represents a query result. In our approach, any intensional components of an iDM graph may be materialized in order to speed-up query processing and/or graph navigation. The discussion on whether to materialize a resource view or not is orthogonal to our model. We are planning to explore that topic as an avenue of future work.

4.3.1 iDM Use-case: Active XML

A special case of querying a remote host is a call to a web service. For this reason iDM can also easily represent XML-specific schemes such as ActiveXML [3]. The basic idea of ActiveXML is to enrich XML documents with calls to web services. Whenever a web service is called, the result of that call will be inserted into the XML document. For instance, consider the ActiveXML document:

```
<dep>
  <sc>web.server.com/GetDepartments()</sc>
</dep>
```

The `<sc>` element contains the web service call. When that web service is executed, its result is inserted into the XML document:

```
<dep>
   <sc>web.server.com/GetDepartments()</sc>
   <deplist>
      <entry>
         <name>Accounting</name>
      </entry>
      ...
   </deplist>
</dep>
```

To instantiate Active XML documents in iDM, it suffices to define a special subclass AXML of resource view class xmlelem with

$$\gamma_i^{\mathsf{AXML}} = \left(\varnothing, \langle V_j^{\mathsf{sc}}[, V_k^{\mathsf{scresult}}]\rangle\right).$$

Here V_j^{sc} contains the call to the web service and $[, V_k^{\mathsf{scresult}}]$ is an optional entry that is only part of the group component if the web service was called.

It is important to note that iDM, in contrast to ActiveXML, also supports graph data and is not restricted to XML documents. Moreover, the pub/sub features of Active XML can also be instantiated in iDM. However, due to space constraints the details are not presented here.

4.4 Infinite Components

Infinite components may occur in two different components of a resource view. First, the content component χ_i may be infinite (see Definition 1). Prominent examples of infinite content are media streams such as audio and video streams, e.g. Real, QuickTime, etc. These may be modelled as infinite sequences of symbols delivered by a content component χ_i. Second, the group component γ_i may also contain a set and/or a sequence of resource views that is infinite (see Definition 1). Prominent examples of infinite group components are data streams (see Section 3.4), publish/subscribe or information filter message notifications [15], and email. In the following, we will discuss why email may be considered infinite.

4.4.1 iDM Use-case: EMail

Consider the example of an email repository. An email repository subscribes to an infinite stream of messages, e.g. all messages containing `jens.dittrich@inf.ethz.ch` in at least one of the TO, CC, or BCC fields. The email server keeps a window on the incoming messages. Users may delete or add messages from/to that window. Using iDM we have two options to model email:

Option 1: (Model the State). We may model the **state** of the INBOX. That state is finite. It can be modelled as

$$\gamma_i^{\mathsf{INBOX\ State}} = \left(\varnothing, \langle V_{\mathsf{q}_1}^{\mathsf{message}}, \ldots, V_{\mathsf{q}_n}^{\mathsf{message}}\rangle\right)$$

where $V_{\mathsf{q}_1}^{\mathsf{message}}, \ldots, V_{\mathsf{q}_n}^{\mathsf{message}}$ represent the messages window currently available in the INBOX, i.e. the current state of the INBOX. Note that for this option the state of that INBOX may be retrieved multiple times.

Option 2: (Model the Stream). Alternatively, we may model the **stream** of incoming messages itself bypassing the state window offered by the email server. This could be achieved as follows:

$$\gamma_i^{\mathsf{INBOX\ message\ stream}} = \left(\varnothing, \langle V_{\mathsf{q'}_1}^{\mathsf{message}}, \ldots, V_{\mathsf{q'}_n}^{\mathsf{message}}\rangle_{n\to\infty}\right),$$

Here, $V_{\mathsf{q'}_1}^{\mathsf{message}}, \ldots, V_{\mathsf{q'}_n}^{\mathsf{message}}$ represent all messages routed to address `jens.dittrich@inf.ethz.ch` during the lifetime of that address. As the stream has no state, messages delivered by the stream cannot be retrieved a second time.

Both options may make sense depending on the application. For instance, Option 1 may be used in those cases where the user uses multiple email clients to read email. In that case, Option 1 could observe the email repository of a user without removing any emails from the server. In contrast, Option 2 is useful in those cases where iDM is used as the single point of access to the user's email repository. In that case, emails streamed to the user would be delivered to the client and removed from the server immediately. No other client would be able to see those messages.

To offer maximum flexibility, our implementation of iDM in `iMeMex` supports both options. We may either represent the state (Option 1) or real data streams providing infinite group components (Option 2). In addition, if we are not able to obtain a real data stream, we may convert a state into a pseudo data stream using a generic polling facility. In the iDM graph that pseudo data stream is then represented as an infinite message stream[5].

4.4.2 Implementing Streams: Need to Push

In order to efficiently support stream processing, any system implementing iDM graphs has to provide push-based protocols [21]. Our current implementation of iDM in `iMeMex` already supports

[5] Interestingly, several popular email services such as POP and IMAP servers do not support the second option. The same applies to RSS/ATOM servers. RSS/ATOM is a method useful for publishing a sequence of messages. However, RSS/ATOM is implemented by simply publishing an XML document containing those messages on a web server. Clients do not get any notifications on changes of that document. For this reason, clients have to poll the server for updates regularly.

push-based operators. Our push-operators may register for changes on any of the components of a resource view. Incoming change events on any resource view, such as a new email message or a new tuple on a data stream, will then be passed to all subscribed push-operators. They will process those events immediately. Like that data-driven stream processing in the spirit of specialized data stream management systems (DSMS) [1] is enabled.

5. iMeMex ARCHITECTURE

This section gives an overview of the implementation of iDM which builds the foundation of the *iMeMex Personal Dataspace Management System (PDSMS)*.

The core idea of iMeMex is to introduce a logical layer that abstracts from underlying subsystems and data sources such as file systems, email servers, network shares, iPods, RSS feeds, etc. This expands on the visions presented in [16] and [20]. We term that logical layer *Resource View Layer*. Figure 4 depicts that layer and its implementation in the iMeMex PDSMS.

iMeMex contains two important sublayers: (1) iQL Query Processor and (2) Resource View Manager.

5.1 iQL Query Processor

The main task of the *iQL Query Processsor* is to parse incoming iQL queries and to create query plans for them. Our current implementation is based on rule-based query optimization. Cost based optimization will be explored as another avenue of future work.

In order to query iDM, we have developed a simple query language termed *iMeMex Query Language (iQL)* that we use to evaluate queries on a resource view graph. Our primary design goal was to have a language that can be used by an *end-user*, i.e. a language that is at the same time *simple and powerful*. For this reason we decided to design our language as an extension to existing IR keyword search interfaces as used by Google and other search engines. Like that users may benefit from our language using only a minimal learning effort. At the same time advanced users may use the same language and interfaces to execute more advanced queries.

One may argue that XML query languages such as XPath and XQuery could be extended to work on top of iDM. The main reasons we did not do this, however, were that XPath and XQuery are too complex to be offered as an end-user language for personal information management. Neverheless, XPath 2.0 has some interesting features. Some of those we have adopted for our language iQL such as path expressions and predicates on attributes. In that respect, our language is close in spirit to NEXI [41]. In contrast to NEXI, however, iQL will include features important for a PDSMS, such as support for updates. A full specification of iQL is beyond the scope of this paper and will be detailed in a separate paper.

In order to get an idea of the expressiveness of our language the following list presents some example iQL queries:

`"Donald Knuth"`: returns all resource views containing the phrase "Donald Knuth" in their content component.

`"Donald" and "Knuth"`: returns all resource views containing both keywords "Donald" and "Knuth" in their content component.

`[size > 42000 and lastmodified < yesterday()]`: returns those resource views having a tuple component attribute greater than 42000 and a lastmodified date before yesterday.

`//Introduction[class="latex_section"]`: returns resource views named "Introduction" and of resource view class "latex_section".

`//PIM//Introduction[class="latex_section"]`: returns every resource view named "Introduction" of class "latex_section" that is indirectly related to a resource view named "PIM".

`//PIM//Introduction[class="latex_section" and "Mike Franklin"]`: returns all resource views named "Introduction" of class "latex_section" that are indirectly related to a resource view named "PIM". All returned results have to contain the phrase "Mike Franklin" in their content component (solves Example 1 from the Introduction).

`//OLAP//[class="figure" and "Indexing time"]`: first, selects resource views that are indirectly related to a resource view named "OLAP". In addition, all results have to be of resource view class "figure" and have to contain the phrase "Indexing time" in their content component (solves Example 2 from the Introduction).

Our current implementation of iQL in iMeMex also supports user-defined joins and graph branching operations. As ongoing work, we are extending iQL to support search over all resource view components and ranking of query results.

5.2 Resource View Manager

The *Resource View Manager (RVM)* is the central instance to managing resource views. It consists of four major components: (1) Data Source Proxy, (2) Content2iDM Converters, (3) Replica&Indexes Module, and (4) Synchronization Manager.

(1) The **Data Source Proxy** provides connectivity to the different types of subsystems. It contains a set of *Data Source Plugins* that represents the data from the different subsystems as an initial iDM graph. Currently we provide plugins for file systems, IMAP email servers and RSS feeds.

(2) The **Content2iDM Converter** further enriches the iDM graph provided by the data source proxy. This is achieved by converting content components to iDM subgraphs that reflect the structural information. Currently we provide converters for XML and LaTeX.

(3) The **Replica&Indexes Module** consists of four parts that reflect the four components of a resource view (name, tuple, content, and group component replicas and indexes). In addition, this module contains a **Resource View Catalog**. All resource views managed are registered in that catalog. For each resource view component we may then choose to insert it into a *replica* and/or an *index*. Infinite group components are managed using a stream window.

A **Replica** creates a copy of a component inside the RVM. For instance, one strategy could be to replicate the *group components* of all resource views that were retrieved from remote data sources. As a consequence queries referring to the group component of resource views can then be executed exploiting the replicas only. This avoids time consuming lookups at the remote data source. As replication may require additional disk and memory space, there is a general trade-off between *data versus query shipping* [32] that has to be considered when creating replication strategies.

An **Index** creates specialized data structures to speed up look-up times. For instance, for *tuple component indexes* hash-tables or B^+-trees can be used to provide efficient access. For textual *content component indexes*, search engines based on inverted keyword lists are state-of-the-art. Note that indexing does not necessarily include replication. For instance, an inverted keyword list is a content component index. However, that index is not able to return the original content component that was used to build the index. This means, it is not at the same time a content component replica. Further note that content indexes are not restricted to text indexes. An example of that is a content index that uses histogram information to index pictures based on image similarity [6].

(4) The **Synchronization Manager** observes all registered data sources for updates. When a new data source is registered at the RVM, the Synchronization Manager will analyze the data found on that data source and send each resource view definition to the Replica&Indexes Module. Depending on the current replication

and indexing strategy the Replica&Indexes Module will then trigger replication and indexing for the newly created resource views. The Synchronization Manager will also poll the data sources regularly to synchronize the catalog, replicas and indexes for updates that were done bypassing the RVM layer. Furthermore, if the data sources support notification events, the Synchronization Manager will subscribe to these notifications. As a consequence, all updates performed on that data source will then be immediately considered by the Synchronization Manager and by the Replica&Indexes Module. For instance, our current implementation is able to subscribe to file events of the hpfs file system created by Mac OS X.

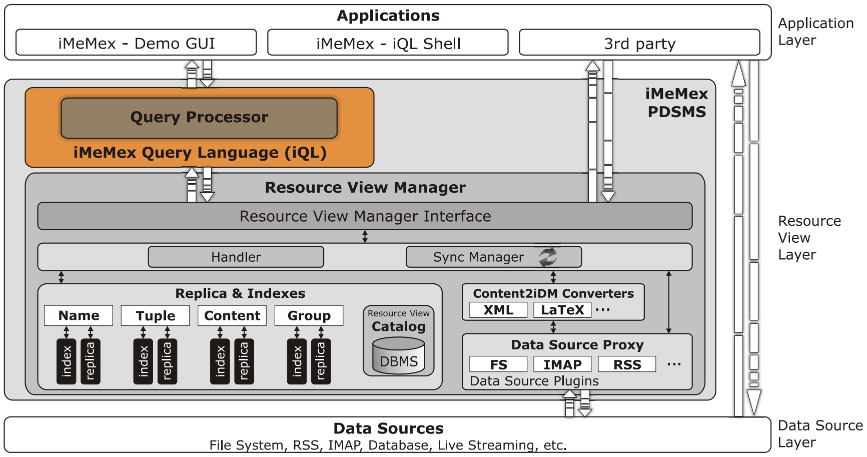


Figure 4: `iMeMex` Architecture

6. RELATED WORK

Personal Information Management is an area broad in scope and we review below several contributions related to our work. We divide them into four categories: data models, operating systems, structure extraction and PIM systems. For each of these categories, we discuss how iDM distinguishes itself from earlier approaches.

Data Models. The relational model has been proposed as a means to provide universal access to data [9]. In the context of information integration, several works have used the relational model to provide a global, data source independent view of data [33, 5]. The Information Manifold [33], for example, is based on the idea of a global schema on top of which the data sources may be expressed as relational views. Object-oriented systems [8] augment relational approaches to represent complex data types. In [40], the authors present the Rufus system, which provides an object based data model for desktop data. The system imports desktop data into a set of pre-defined classes and provides querying capabilities over those classes. All of these previous approaches are based on a schema-first modeling strategy that is not well suited for personal information. In contrast, our data model is in line with the goals recently presented in [20], which claim that future information systems must be capable of offering a wide range of modeling strategies, such as *schema-later* or even *schema-never*.

Semi-structured approaches [37, 45] have been proposed as an alternative to self-describe data and thus eliminate the need to provide pre-defined schemas. Currently, the most prominent semi-structured model is XML [45]. XML is typically associated with a specific serialization for data exchange, which means that it becomes harder to represent data in its data model from a purely logical standpoint, as advocated in [2]. Further, XML is not capable of representing graph, intensional and infinite data without using add-on techniques. However, the former types of data are key in personal information management and therefore fully supported by iDM. Some approaches have tried to extend XML in order to cope with these limitations. Colorful XML [28] proposes a data model in which an XML document is extended with multiple hierarchies, distinguished by node coloring. ActiveXML [36, 3] represents intensional data in XML documents by embedding web service calls. In sharp contrast, iDM offers a logical model that represents a wide range of available heterogeneous personal information and integrates all of the above approaches into one generic framework. For instance, in Section 4.3.1 we show how ActiveXML may be modeled as a use case of iDM. Some recent works have developed keyword search techniques considering graph data models to represent heterogeneous data [30, 24]. The search techniques developed are orthogonal to our approach and may also be used on top of iDM.

Operating Systems. Recently, modern operating systems have been amended to provide full-text *search* capabilities. Examples of such systems are Google Desktop [23], MS Desktop Search [42], and Apple Spotlight [4]. These systems use simple data models based on sequences of words. Unlike in iDM, the structural information available within files is not exploited for queries.

The linux add-on file system Reiser FS [38] blurs the distinction among files and folders by letting files behave as folders in special circumstances, e.g. by displaying file attributes as files. Microsoft WinFS [44], now discontinued[6], proposed to base storage of personal information in a relational database. Data was represented in WinFS in an item data model which is a subset of the object-oriented data model. Like in previous object-oriented approaches for personal information [40], WinFS employed a schema-first data modeling strategy. This problem was addressed in WinFS in a brute-force fashion, i.e., by shipping the system with a wide array of pre-defined schemas for personal information that may then be extended by application developers. In sharp contrast to that, in iDM there is no need to import generally available data as a variety of data models may already be expressed as constrained versions of iDM (see Section 3). That allows a system based on iDM to be useful from the start — with investment on schemas being made only if necessary [20]. In addition, from a systems perspective, WinFS represented a platform-specific solution restricted to a vendor specific set of devices. Our approach is totally different from that as the iDM-based `iMeMex` PDSMS already works with a larger set of operating systems including Linux and other Unixes, Windows, and Mac OS X (see Section 5). At the same time, `iMeMex` is able to seamlessly integrate into all of the above operating systems [16].

Structure Extraction. Approaches to the problem of structure extraction are *automatic and semi-automatic*. Some techniques for *automatic* structure extraction have a pre-defined schema to which entity references are matched, e.g. reference reconciliation [18] and email classification [10, 27], while others seek to detect categories on the data, e.g. topic and social network analysis [35, 12]. These techniques are orthogonal to our approach, as they provide means to extract structure from data; derived information may be represented in a variety of data models, including iDM.

Fully automatic structure extraction is a long-term goal. Therefore, *semi-automatic* extraction approaches [26] try to increase ac-

[6] The downloadable beta as well as all other preliminary information about WinFS were recently removed from its web-site [44].

curacy by incorporating user interference. Hubble [34] is a system that allows users to organize and categorize XML files into dynamically generated folders using complex XQuery expressions. In contrast to iDM, Hubble is restricted to extensional XML files and does not suppport infinite data. Moreover, iDM provides a logical data model to unify the XML file's elements hierarchy and the outside folder hierachy.

PIM Systems. Systems such as SEMEX [17] and Haystack [31] enable users to *browse by association*. The information in the desktop is extracted and imported into a repository that encodes pre-defined schemas about high-level entities like persons, projects, publications, etc. The rich relationships among such personal data items call for a data model that may represent arbitrary graph data and not only trees or DAGs. Lifestreams [22] completely abandons the files&folders paradigm and bases the organization of information on a timeline. MyLifeBits [7] allows visualizations to be built on top of a unified store for personal information. All information, including resource content, is stored in a relational database. Queries are represented in these systems as collections whose content is calculated on demand. Such use-cases of intensionally defined data may also be modeled in iDM (see Section 4.3).

7. EVALUATION

The goal of this experimental evaluation is to show that iDM can be efficiently implemented in a real PDSMS. We present results based on an initial implementation of iDM in iMeMex. We report indexing times and sizes for an actual personal dataset. Further, we execute different classes of queries over iDM and report query response times.

7.1 Setup

We performed our experiments on an Intel Pentium M 1.8 GHz PC with 1 GB of main memory and an IDE disk of 60 GB. We used MS Windows XP Professional SP2 as its operating system and a volume with NTFS. iMeMex is implemented in Java and we used Sun's hotspot JVM 1.4.2_08-b03. The JVM was configured to allocate at most 512 MB of main memory. The Resource View Catalog is implemented on top of Apache Derby 10.1 and full-text indexes use Apache Lucene 1.4.3. The full-text of all text based content and PDF content is indexed. Further, we have implemented converters that take content components in XML or in LaTeX formats and generate resource view graphs as shown in Section 2.

Many authors have observed that there is a lack of benchmarks to evaluate PIM approaches, e.g. [29, 43]. Thus, we have decided to use a real personal dataset to evaluate the efficiency of our iDM implementation. Our dataset consists of the personal files and emails of one of the authors of this paper. All files are kept on the same computer in which iMeMex is run; emails are kept on a remote server and are accessed via the IMAP protocol.

The characteristics of the data set used are shown in Table 2. We show each data source and the total number of resource views extracted from the data source. The total number of resource views may be broken into the number of resource views obtained by representing base items of the data source (i.e., files&folders for the filesystem; emails, folders and attachments for email), the number of resource views extracted from XML content, and the number of resource views extracted from LaTeX content. We also show the total storage requirements of the items stored in the data sources.

As we may observe in Table 2, the largest portion of the data is stored locally on the filesystem. Most of the resource views present on the filesystem data source are obtained from the content of XML and LaTeX files. These views were obtained from 47 XML documents and 282 LaTeX documents. In the email data source, the number of resource views extracted from XML or LaTeX documents is relatively smaller, as in this dataset these documents are not commonly exchanged as attachments to email messages. We have 13 XML documents and 7 LaTeX documents as attachments to email messages. When both data sources are taken together, the number of resource views derived from base items greatly surpasses the number of base items.

7.2 Results

We report results on two experiments using the dataset described in the previous section. In the first experiment, we show iMeMex index and replica creation times and sizes; in the second experiment, we evaluate query performance when exploiting iDM to run a set of queries over structured and unstructured desktop data.

Indexing. iMeMex offers a set of index and replica implementations to speed-up query processing. In our current implementation, we use the following structures:

1. **Name Index & Replica**: an Apache Lucene full-text index that also stores the values of the resource views' name components.
2. **Tuple Index & Replica**: a replica of all resource views' tuple components. iMeMex keeps this replica in-memory and an auxiliary sorted index structure is also kept in-memory. This index structure is based on vertical partitioning [11].
3. **Content Index**: an Apache Lucene full-text index on text extracted from content components, if possible. This index is not a replica, i.e., the original content is not saved in the index.
4. **Group Replica**: a replica of all resource views' group components. iMeMex keeps this replica in-memory.

For the remainder, we refer to indexes and replicas simply by the term *indexes*, as their use in this evaluation is to improve query processing time. In our initial implementation, all resource views present in the data source are registered in the resource view catalog and their components are indexed in the structures described above.

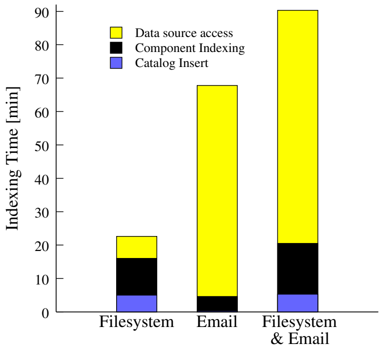


Figure 5: Indexing times [min]

Table 3 shows the resulting *index sizes* for the personal dataset. Although the original dataset occupies around 4 GB, we observe that the net input data size is of about 255 MB. The net input data size is obtained by excluding the content size of those resource views whose content could not be converted to a textual representation (e.g. image formats). That content was therefore not given as input to the content index. As we may notice on Table 3, the total size for all indexes is 67.5% of the net input data size. Most of that total index size is taken by the full-text index on content.

The total *indexing time* for the personal dataset described in Section 7.1 is shown in Figure 5. For each data source, we report the time necessary to register metadata in the Resource View Catalog (Catalog Insert), the time spent inserting data in our index structures (Component Indexing) and the time spent to obtain the data

Data Source	Total Size (MB)	# of Resource Views						
		Base Views			Derived Views			Total
		Files&Folders	Email	Total	XML	LaTeX	Total	
Filesystem	4,243	14,297	0	14,297	117,298	11,528	128,826	143,123
Email / IMAP	189	0	6,335	6,335	672	350	1,022	7,357
Total	**4,435**	**14,297**	**6,335**	**20,632**	**117,970**	**11,878**	**129,848**	**150,480**

Table 2: Characteristics of the personal dataset used in the evaluation

Data Source	Net Input Data Size (MB)	Index Sizes (MB)					
		Name	Tuple	Content	Group	RV Catalog	Total
Filesystem	212.3	12.5	11.5	113.0	3.3	24.4	**164.7**
Email / IMAP	43.1	0.4	1.8	5.0	0.2	0.4	**7.8**
Total	**255.4**	**12.9**	**13.3**	**118.0**	**3.5**	**24.8**	**172.5**

Table 3: Index sizes for the personal dataset

from the underlying data sources (Data Source Access). For the filesystem, the total indexing time is about 22 min. Roughly half of that time is spent on inserting information on index data structures, while the remaing time is distributed among catalog maintenance and the actual scanning of the underlying filesystem. The indexing of email takes about 68 min and the time is dominated by data source access. The catalog maintenance time during email indexing is negligible, as the email data source contains only a small number of resource views. Overall, the influence of indexing the remote email data source on the total indexing time is significant, as most of the time is spent with data source access.

	iQL Query expression	**# of Results**
Q1	“database”	941
Q2	“database tuning”	39
Q3	[size > 420000 and lastmodified < @12.06.2005]	88
Q4	//papers//*Vision/*[“Franklin”]	2
Q5	//VLDB200?//?onclusion*/*[“systems”]	2

Q6	union(//VLDB2005//*[“documents”],
	//VLDB2006//*[“documents”])
		31

Q7	join(//VLDB2006//*[class=“texref”] as A,
	//VLDB2006//*[class=“environment”]//figure*
	as B, A.name=B.tuple.label)
		21

Q8	join (//*[class = “emailmessage”]//*.tex as A, //papers//*.tex as B, A.name = B.name)	16

Table 4: iQL queries used in the evaluation

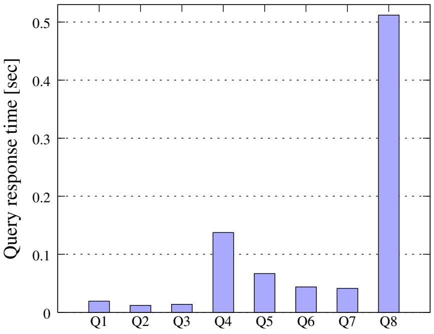


Figure 6: Query response times for queries Q1–Q8 [sec]

Query Processing. We focus our evaluation on response times observed for a set of queries over the desktop dataset used. The queries evaluated are shown in Table 4, along with their number of results. Figure 6 shows the observed response times for the eight queries evaluated. We report all execution times with a warm cache, i.e., each query is executed several times until the deviation on the average execution time becomes small. As we may notice, most queries evaluated (Q1-Q7) are executed in less than 0.2 seconds. The only exception is Q8, a join query that includes information from different subsystems (email, filesystem) and takes about 0.5 seconds. The reason is that, after fetching the data via index accesses, our query processor obtains indirectly related resource views by forward expansion. For Q8, that causes the processing of a large number of intermediate results when compared to the final result size. In order to provide even better response times in such situations, we plan to investigate alternative processing strategies such as backward or bidirectional expansion [30]. Nevertheless, in the HCI community it is argued that a response time of less than 1 second should be the goal for every computer system [39]. Thus, we conclude that for a sample of meaningful queries over real desktop data the implementation of iDM in iMeMex allows for query processing with interactive response times.

8. CONCLUSIONS

Personal Information Management has become a key necessity of almost everybody. Several techniques and systems have been proposed for various scenarios and operating systems. Considerable progress has been made in the PIM area in the recent past. At the same time, it has become clear that what is missing is a unified approach to PIM that is able to represent *all* personal information in a single, powerful and yet simple data model. This paper has proposed the iMeMex Data Model (iDM) as a unified and versatile solution. The major advantages of our approach are: (1) iDM clearly differentiates between the logical data model and its physical representation, (2) iDM is powerful enough to represent XML, relations, files&folders and cyclic graphs in a single data model, (3) iDM is able to represent the structural contents inside files as part of the same data model, (4) iDM is powerful enough to represent extensional data (base facts), intensional data (e.g. ActiveXML), as well as infinite data (content and data streams), (5) iDM enables a new class of queries that are not available with state-of-the-art PIM tools.

iDM builds the foundation of the *iMeMex Personal Dataspace Management System*. We have demonstrated that iDM can be efficiently implemented in such type of system offering both quick indexing times and interactive query response times.

We point out below some issues relevant to personal dataspace management that are orthogonal to our model, but which are easier to tackle once a data model like iDM is in place:

1. **Versioning.** A PDSMS keeps track of all changes made to the dataspace. As with classical versioning techniques, logically, each change creates a new version of the whole dataspace. iDM allows the representation of the entire dataspace of a user in one model. Thus, the implementation of versioning is strongly

simplified.

2. **Lineage.** Data lineage refers to keeping the history of all data transformations that originated a given resource view. For example, when a user copies a file into another and then modifies the new file, the system should keep for the new resource view the information about its provenance. With a unified model such as iDM, it is possible to keep lineage information across data sources and formats.

The exploration of these issues is part of ongoing and future work in the `iMeMex` Personal Dataspace Management System. We plan to present an updated and extended description of `iMeMex`'s architecture as a separate paper. We will also develop a full specification of iQL and investigate algorithms for query processing and cost-based query optimization. In addition, we are planning to extend our system to enable networks of P2P instances. Finally, we are planning to explore PIM applications such as reference reconciliation and clustering on top of the `iMeMex` platform.

Acknowledgements

First of all, we would like to thank the anonymous reviewers for their insightful comments. Further, we would like to thank our colleagues Donald Kossmann, Peter Fischer, Rokas Tamosevicius and Shant Karakashian for comments on earlier versions of this paper. We also thank Shant Karakashian for providing the LaTeX2iDM converter. In addition, we thank Olivier Girard and Marco Steybe for their help on implementing iQL and the different index structures, respectively.

9. REFERENCES

[1] D. J. Abadi et al. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[2] S. Abiteboul. On Views and XML. In *PODS*, 1999.

[3] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *ACM SIGMOD*, 2003.

[4] `http://www.apple.com/macosx/features/spotlight`. Apple Mac OS X Spotlight.

[5] 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.

[6] J. Ashley et al. The Query By Image Content (QBIC) System (Demo Paper). In *ACM SIGMOD*, 1995.

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

[8] R. G. G. Cattell et al., editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.

[9] E. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6), 1970.

[10] W. Cohen, V. Carvalho, and T. Mitchell. Learning to Classify Email into "Speech Acts". In *EMNLP*, 2004.

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

[12] A. Culotta, R. Bekkerman, and A. McCallum. Extracting social networks and contact information from email and the Web. In *Conf. on Email and Anti-Spam (CEAS)*, 2004.

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

[14] J.-P. Dittrich. iMeMex: A Platform for Personal Dataspace Management. In *SIGIR PIM Workshop*, 2006. To appear.

[15] J.-P. Dittrich, P. M. Fischer, and D. Kossmann. AGILE: Adaptive Indexing for Context-Aware Information Filters. In *ACM SIGMOD*, 2005.

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

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

[18] X. Dong, A. Halevy, J. Madhavan, and E. Nemes. Reference Reconciliation in Complex Information Spaces. In *ACM SIGMOD*, 2005.

[19] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2000. Third Edition.

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

[21] M. J. Franklin and S. B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *ACM SIGMOD*, 1998.

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

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

[24] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In *VLDB*, 2005.

[25] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index Selection for OLAP. In *IEEE ICDE*, 1997.

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

[27] Y. Huang, D. Govindaraju, T. Mitchell, V. Carvalho, and W. Cohen. Inferring Ongoing Activities of Workstation Users by Clustering Email. In *Conf. on Email and Anti-Spam (CEAS)*, 2004.

[28] H. V. Jagadish, L. V. S. Lakshmanan, M. Scannapieco, D. Srivastava, and N. Wiwatwattana. Colorful XML: one hierarchy isn't enough. In *ACM SIGMOD*, 2004.

[29] 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.

[30] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB*, 2005.

[31] 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.

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

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

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

[35] A. McCallum, A. Corrada-Emmanuel, and X. Wang. The Author-Recipient-Topic Model for Topic and Role Discovery in Social Networks: Experiments with Enron and Academic Email. In *TR Uni. of Massachusetts Amherst*, 12/2004.

[36] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. D. Ngoc. Exchanging Intensional XML Data. In *ACM SIGMOD*, 2003.

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

[38] `http://www.namesys.com`. Reiser FS.

[39] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Computing Surveys*, 16(3):265–285, 1984.

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

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

[42] `http://www.microsoft.com/windows/desktopsearch`. Windows Desktop Search.

[43] G. Weikum. An Experiment: How to Plan it, Run it, and Get it Published (Presentation). In *CIDR*, 2005.

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

[45] XQuery 1.0 and XPath 2.0 Data Model (XDM), 2005. http://www.w3.org/TR/xpath-datamodel/.

[46] XML Information Set (Second Edition), 2004. http://www.w3.org/TR/xml-infoset.

[47] `http://www.imemex.org`. iMeMex project web-site.